Widget Reference (R – S) 9

RectButton Widget

Class

Class Name: `RectButton`
Class Pointer: `rectButtonWidgetClass`

Ancestry

Core-Primitive-Button-RectButton

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/RectButton.h>
```

Description

The RectButton is a toggle button that can be either *set* or *unset* and is designed to be a choice item in a one-of-many or several-of-many choice object. When the user presses SELECT on the button, its state will toggle. The state of the button can also be set programmatically.

Components

The figure below shows several OPEN LOOK compliant buttons, in normal, default, and current states (two versions).

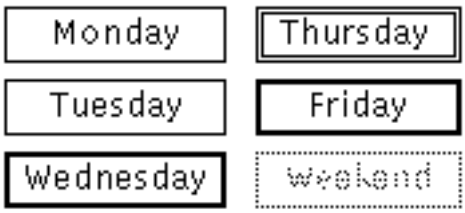

Figure 9-1 RectButton Rectangular Buttons

The RectButton widget consists of a Label surrounded by a rectangular border. The border can change to reflect that the button may be a default of several buttons (double border), or represents a current state of an object (thick border), or represents a current state of one of several objects with different states (dimmed border).

The RectButton widget is not intended to be used alone, but rather as a child of either an Exclusives composite widget (to implement one-of-many choice selection—see "Exclusives Widget" on page 277) or a Nonexclusives composite widget (to implement several-of-many choice selection—see "Nonexclusives Widget" on page 428). If the RectButton is created as a child of a different composite widget, proper behavior is not guaranteed.

The following descriptions of using the RectButton widget in a menu assume that the RectButton widget is a child of an Exclusives or Nonexclusives widget, which is a descendant of a MenuShell widget.

Popup Menu RectButtons

Entering a RectButton while MENU is depressed changes the appearance of the button from unset to set state or vice versa, to reflect the state the button would be in if MENU were released. Releasing MENU toggles the state associated with the button. Leaving the button before releasing MENU restores the original state appearance and does not toggle the button.

Non-Popup Menu RectButtons

Clicking SELECT on a RectButton toggles the state associated with it. Pressing SELECT, or moving the pointer into the button while SELECT is pressed, changes the border from unset to set state or vice versa, to reflect the state the button would be in if SELECT were released. Releasing SELECT toggles the state. Moving the pointer off the button before releasing SELECT restores the state appearance and does not toggle the button.

If the button is in a stay-up menu, clicking or pressing MENU works the same as SELECT. If the button is not in a stay-up (or popup) menu, clicking or pressing MENU does not do anything; the event is passed up to an ancestor widget.

Coloration

For 3D, RectButton coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1. `XtNfontColor` is used to draw the label.

For 2D, `XtNbackground` and `XtNfontColor` are used to render the RectButton as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

If the toolkit resource `XtNmouseless` is set to TRUE and the toolkit resource `XtNinputFocusFeedback` is set to OL_INPUT_FOCUS_COLOR, then the background of the RectButton will be drawn with the value of `XtNinputFocusColor` when the widget receives input focus. However, if `XtNinputFocusColor` is the same as `XtNbackground`, then the RectButton inverts `XtNfontColor` and `XtNbackground`. Once the input focus leaves the widget, the original coloration is restored.

Keyboard Traversal

The default value of the `XtNtraversalOn` resource is TRUE.

The RectButton widget responds to the following keyboard navigation keys:

- NEXTFIELD moves to the next traversable widget in the window
- PREVFIELD moves to the previous traversable widget in the window

- MOVEUP moves to the RectButton above the current widget in the Nonexclusives or Exclusives composite
- MOVEDOWN moves to the RectButton below the current widget in the Nonexclusives or Exclusives composite
- LEFT moves to the RectButton to the left of the current widget in the Nonexclusives or Exclusives composite
- MOVERIGHT moves to the RectButton to the right of the current widget in the Nonexclusives or Exclusives composite
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application

The RectButton will respond to the SELECTKEY by acting as if the SELECT buttons had been clicked.

Keyboard Mnemonic Display

The RectButton widget displays its mnemonic accelerator as part of its label. If the mnemonic character is in the label, then that character is highlighted according to the value of the `XtNshowMnemonics` toolkit resource. If the mnemonic character is not in the label, it is displayed to the right of the label in parenthesis and highlighted according to the value of the `XtNshowMnemonics` resource.

If label truncation is necessary, the mnemonic displayed in parenthesis is truncated as a unit.

Keyboard Accelerator Display

The display of keyboard accelerators is controlled by the toolkit resource `XtNshowAccelerators`. When the value of `XtNshowAccelerators` is OL_DISPLAY, the RectButton widget displays the keyboard accelerator as part of its label. The string in the `XtNacceleratorText` resource is displayed to the right of the label (or mnemonic) separated by at least one space. The acceleratorText is right justified.

If label truncation is necessary, the accelerator is truncated as a unit. The accelerator is truncated before the mnemonic or the label.

Resources

Table 9-1 RectButton Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	(calculated)	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	(calculated)	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-2 RectButton Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	SGI
XtNreferenceName	String	NULL	GI

Table 9-2 RectButton Primitive Resources (Continued)

Name	Type	Default	Access
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNunrealizeCallback	XtCallbackList	NULL	SGIO
XtNuserData	XtPointer	NULL	SGI

Table 9-3 RectButton Resources

Name	Type	Default	Access
XtNdefault	Boolean	FALSE	SGI
XtNdim	Boolean	FALSE	SGI
XtNlabel	OlStr	(instance name)	SGI
XtNlabelImage	XImage *	NULL	SGI
XtNlabelJustify	OlDefine	OL_LEFT	SGI
XtNlabelTile	Boolean	FALSE	SGI
XtNlabelType	OlDefine	OL_STRING	SGI
XtNrecomputeSize	Boolean	TRUE	SGI
XtNselect	XtCallbackList	NULL	SGIO
XtNset	Boolean	FALSE	SGI
XtNunselect	XtCallbackList	NULL	SGIO

XtNdefault

Class	Type	Default	Access
XtCDefault	Boolean	FALSE	SGI

Synopsis: Whether the RectButton is the default choice in its immediate shell.

Values: `TRUE/"true"` – This button is the default control and it will indicate as such by displaying a border doubled to two lines.
`FALSE/"false"` – The button is not the default choice.

XtNdim

Class	Type	Default	Access
XtCDim	Boolean	FALSE	SGI

Synopsis: Whether the border of the RectButton visually reflects the state of associated objects.

Values: `TRUE/"true"` – The border is dimmed to show that the RectButton represents the state of one or more of several objects that, as a group, are in different states.
`FALSE/"false"` – The border is not dimmed.

XtNlabel

Class	Type	Default	Access
XtCLabel	OlStr	(instance name)	SGI

Synopsis: The text for the Label.

Values: Any `OlStr` value valid in the current locale.

This resource is ignored if the `XtNlabelType` resource has the value OL_IMAGE.

The RectButton label is colored using the `XtNfontColor` resource.

XtNlabelImage

Class	Type	Default	Access
XtCLabelImage	XImage *	NULL	SGI

Synopsis: The image for the Label.

This resource is ignored unless the `XtNlabelType` resource has the value OL_IMAGE. If the image is of type `XYBitmap`, the image is highlighted when appropriate by reversing the 0 and 1 values of each pixel (that is, by XORing the image data). If the image is of type `XYPixmap` or `ZPixmap`, the image is not highlighted, although the space around the image inside the border is highlighted.

If the image is smaller than the space available for it inside the border and `XtNlabelTile` is FALSE, the image is centered vertically and either centered or left-justified horizontally, depending on the value of the `XtNlabelJustify` resource. If the image is larger than the space available for it, it is clipped so that it does not display outside the border. If the `XtNdefault` resource is TRUE

so that the border is doubled, the space available is that inside the inner line of the border.

XtNlabelJustify

Class	Type	Default	Access
XtCLabelJustify	OlDefine	OL_LEFT	SGI

Synopsis: The justification of the Label within the widget width.
Values: `OL_LEFT/"left"` – The Label is left-justified.
`OL_RIGHT/"right"` – The Label is right-justified.
`OL_CENTER/"center"` – The Label is centered.

XtNlabelTile

Class	Type	Default	Access
XtCLabelTile	Boolean	FALSE	SGI

Synopsis: The tiling of the Label's background.
Values: `TRUE/"true"` – For an image that is smaller than the subobject's background, the label area is tiled with the image to fill the subobject's background.
`FALSE/"false"` – The label is placed as described by the `XtNlabelJustify` resource.

This resource augments the `XtNlabelImage` resource to allow tiling the subobject's background. The `XtNlabelTile` resource is ignored for text labels.

XtNlabelType

Class	Type	Default	Access
XtCLabelType	OlDefine	OL_STRING	SGI

Synopsis: The form that the Label takes.
Values: `OL_STRING/"string"` – The label is text.
`OL_IMAGE/"image"` – The label is an image.

XtNrecomputeSize

Class	Type	Default	Access
XtCRecomputeSize	Boolean	TRUE	SGI

Synopsis: The resize policy of the widget.

Values: `TRUE/"true"` – The RectButton widget will do normal size calculations that may cause its geometry to change, and automatically set the `XtNheight` and `XtNwidth` resources. `FALSE/"false"` – The RectButton widget will leave its size alone; this may cause truncation of the visible image being shown by the RectButton widget if the fixed size is too small, or may cause padding if the fixed size is too large. The location of the padding is determined by the `XtNlabelJustify` resource.

XtNselect

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the widget is selected.

XtNset

Class	Type	Default	Access
XtCSet	Boolean	FALSE	SGI

Synopsis: The current state of the button.
Values: `TRUE/"true"` – The button is set (if the toolkit resource `XtNthreeD` is TRUE, the button will appear pressed-in; otherwise, the border is thickened to indicate the set state).
`FALSE/"false"` – The button is unset.

Simply setting `XtNset` to TRUE with a call to `XtSetValues()` does not issue the `XtNselect` callbacks.

XtNunselect

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the widget is toggled into "unset" mode.

When the `RectButton` is toggled into "unset" mode by the user to make `XtNset` become FALSE, the callbacks specified in this resource are activated.

Simply setting `XtNset` to FALSE with a call to `XtSetValues()` does not issue the `XtNunselect` callbacks.

Border Resource Interactions

The `XtNdim`, `XtNdefault`, and `XtNset` resources can be set independently; however, all these states cannot be reflected in the visual appearance of the RectButton, as shown in the following table.

Table 9-4 RectButton Borders

XtNset	XtNdefault	XtNdim	Border appearance
TRUE	TRUE/FALSE	TRUE	Dimmed
TRUE	TRUE/FALSE	FALSE	Thickened
TRUE	TRUE	TRUE	Open
FALSE	TRUE	FALSE	Open
FALSE	TRUE	TRUE	Normal
FALSE	FALSE	FALSE	Normal

Label Resource Interactions

The `XtNwidth`, `XtNheight`, `XtNrecomputeSize`, and `XtNlabelJustify` resources interact to produce a truncated, clipped, centered, or left-justified label as shown in the following table.

When the label is centered or left-justified, the extra space is filled with the background color of the RectButton widget, as determined by the `XtNbackground` and `XtNbackgroundPixmap` resources. When the label is truncated, a solid-black triangle is inserted to show that part of the label is missing. The triangle requires that more of the label be truncated than would otherwise be necessary. If the width of the button is too small to show even one character with the triangle, only the triangle is shown. If the width is so small that the entire triangle cannot be shown, the triangle is clipped on the right. See the `XtNlabelTile` resource for how it affects the appearance of a label.

Table 9-5 RectButton Label Appearance

XtNwidth	XtNrecomputeSize	XtNlabelJustify	Result
any value	TRUE	any	Just Fits
> needed for label	FALSE	OL_LEFT	Left Justified
> needed for label	FALSE	OL_CENTER	Right Justified
< needed for label	FALSE	any	Trunc▷
XtNheight	**XtNrecomputerSize**	**XtNlabelJustify**	**Result**
any value	TRUE	any	Just Fits
> needed for label	FALSE	any	Centered
< needed for label	FALSE	any	**Clipped**

Activation Types

The following table lists the activation types used by the RectButton.

Table 9-6 RectButton Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MENU	MENU	XtNmenuBtn
OL_MENUDEFAULT	MENUDEFAULT	XtNmenuDefaultBtn
OL_MENUDEFAULTKEY	MENUDEFAULT	XtNmenuDefaultKey
OL_MENUKEY	MENU	XtNmenuKey
OL_MOVEDOWN	DOWN	XtNdownKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	LEFT	XtNleftKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey

Table 9-6 RectButton Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_MOVERIGHT	RIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_MOVEUP	UP	XtNupKey
OL_MULTIDOWN	JUMP DOWN	XtNmultiDownKey
OL_MULTILEFT	JUMP LEFT	XtNmultiLeftKey
OL_MULTIRIGHT	JUMP RIGHT	XtNmultiRightKey
OL_MULTIUP	JUMP UP	XtNmultiUpKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_NEXTWINDOW	NEXTWINDOW	XtNnextWinKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_PREVWINDOW	PREVWINDOW	XtNprevWinKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

Activation types not described in the following list are described in "Common Activation Types" on page 68.

OL_MENU/ OL_MENUKEY

The RectButton only will respond to the OL_MENU and OL_MENUKEY activation types if it is a descendant of a Menu widget. When this is the case, the OL_MENU and OL_MENUKEY will behave as the OL_SELECT and OL_SELECTKEY, respectively.

OL_MENUDEFAULT/ OL_MENUDEFAULTKEY

The OL_MENUDEFAULT and OL_MENUDEFAULTKEY activation types only apply to RectButtons that are descendants of a Menu. These activation types set the RectButton `XtNdefault` resource to TRUE, and change the display of the widget according to the *OPEN LOOK GUI Functional Specification* section "Changing Menu Defaults" in Chapter 15.

OL_SELECT/
OL_SELECTKEY

The activation of the RectButton widget with the SELECT button or key will depend on the parent of the RectButton: either Exclusives or Nonexclusives.

The activation of an Exclusive RectButton is described in the *OPEN LOOK GUI Functional Specification* section "Exclusive Settings" in Chapter 4 and in "Using Menus" in Chapter 15. When the user selects a RectButton in an exclusive setting, the `XtNset` resource will be set to TRUE and the `XtNselect` callback will be called. In addition, the RectButton in the exclusive that was previously set will have the `XtNset` resource changed to FALSE and the `XtNunselect` callback will be called.

The activation of a Nonexclusive RectButton is described in the *OPEN LOOK GUI Functional Specification* section "Nonexclusive Settings" in Chapter 4 and in "Using Menus" in Chapter 15. When the user selects a RectButton in a nonexclusive setting, the state of the `XtNset` resource is reversed. When the `XtNset` resource goes to FALSE, the `XtNunselect` callback is called; when the `XtNset` resource goes to TRUE, the `XtNselect` callback will be called.

See Also

"Exclusives Widget" on page 277,
"FlatExclusives Widget" on page 337,
"FlatNonexclusives Widget" on page 347,
"MenuButton Widget" on page 403,
"Nonexclusives Widget" on page 428,
"OblongButton Widget" on page 464.

9

RubberTile Widget

RubberTile Widget

Class

Class Name: `RubberTile`
Class Pointer: `rubberTileWidgetClass`

Ancestry

Core-Composite-Constraint-Manager-RubberTile

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/RubberTile.h>
```

Description

The RubberTile is a constraint widget that allows an application to lay out its children either vertically or horizontally, and then assign relative weights to each child so that it absorbs a certain percentage of size changes. If the RubberTile is set with a vertical orientation, then the children will be laid out vertically in column, each spanning the width of the RubberTile. If the RubberTile is laid out horizontally, then the children will be laid out horizontally in a row, each ones height spanning the height of the RubberTile. The RubberTile resizes its children according to a weight assigned to each child. If there are three children with weights of 1, 2, and 3, they resize to get 1/6th, 1/3rd, and 1/2 of the available space respectively.

This widget is very useful when laying out panes in a window. For example, if an application requires three panes laid out vertically, and the top pane is not to absorb any height changes, but the two lower panes are to each absorb half of the height changes, then the application can do the following: create a RubberTile with `XtNorientation` set to OL_VERTICAL, then create each child, assigning each the appropriate `XtNweight` constraint resource value:

top pane	->	`XtNweight` = 0
middle pane	->	`XtNweight` = 1
bottom pane	->	`XtNweight` = 1

top pane's size change percentage = 0 / (0 + 1 + 1) = 0%
middle pane's size change percentage = 1 / (0 + 1 + 1) = 50%
bottom pane's size change percentage = 1 / (0 + 1 + 1) = 50%

RubberTiles do not enforce any weighting on their children when the widget is created, only when it is resized. During creation, for a horizontal layout, the initial width of the RubberTile widget will be the sum of the width of the children and the initial height will be that of the tallest child. For a vertical layout, the initial height of the RubberTile widget will be the sum of the individual heights and the initial width will be that of the widest child.

An individual RubberTile only supports a single dimensional array, vertical or horizontal. However, an application can produce a two dimensional effect with a matrix of RubberTile widgets. For example, to create the effect of a 2 × 4 matrix, two top level RubberTiles could manage four RubberTiles each.

Coloration

The diagram illustrates the resources that affect RubberTile coloration.

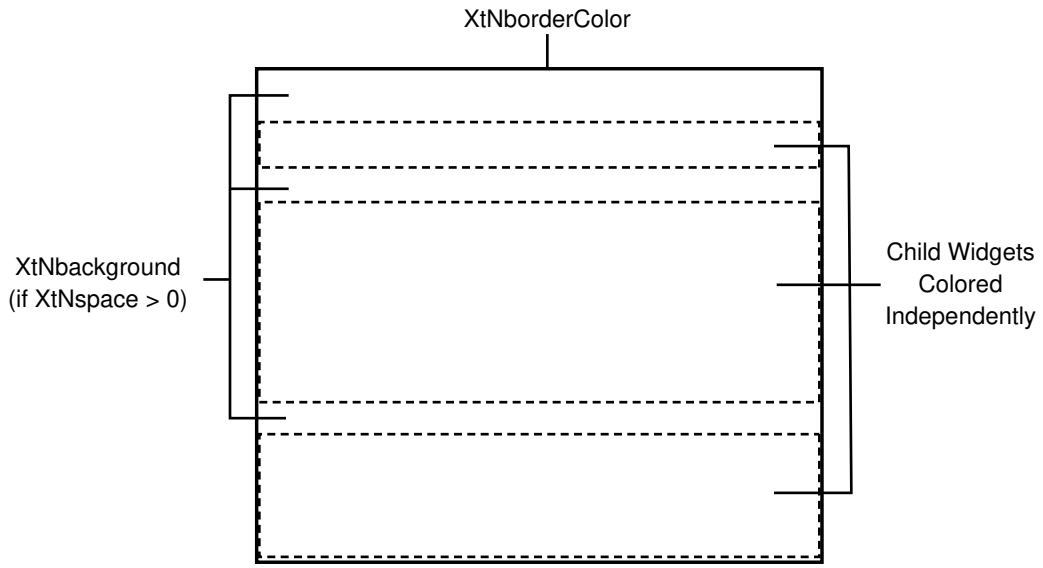

Figure 9-2 RubberTile Coloration

9

RubberTile Widget Resources

Table 9-7 RubberTile Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-8 RubberTile Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 9-9 RubberTile Manager Resources

Name	Type	Default	Access
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNinputFocusColor	Pixel	Red	SGID

Table 9-9 RubberTile Manager Resources (Continued)

Name	Type	Default	Access
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNunrealizeCallback	XtCallbackList	NULL	SGIO
XtNuserData	XtPointer	NULL	SGI

Table 9-10 RubberTile Resources

Name	Type	Default	Access
XtNorientation	OlDefine	OL_VERTICAL	SGI

Each child widget attached to the RubberTile composite widget is constrained by the resources in the following table. These resources become resources for each child widget and can be set and read just like any other resources defined for the child.

Table 9-11 RubberTile Constraint Resources

Name	Type	Default	Access
XtNrefName	String	NULL	SGI
XtNrefWidget	XtPointer	NULL	SGI
XtNspace	Dimension	0	SGI
XtNweight	Dimension	1	SGI

XtNorientation

Class	Type	Default	Access
XtCOrientation	OlDefine	OL_VERTICAL	SGI

Synopsis: The orientation of the widget.

Values: `OL_VERTICAL/"vertical"` – Arrange the child widgets as a single vertical column.

`OL_HORIZONTAL/"horizontal"` – Arrange the child widgets as a single horizontal row.

XtNrefName

Class	Type	Default	Access
XtCRefName	String	NULL	SGI

Synopsis: The reference widget by name.
Values: The name of the child of the RubberTile.

When a child is positioned by the RubberTile widget, it is placed geometrically after (below or right of) the widget referenced by `XtNrefName` or `XtNrefWidget`.

The reference named by this resource will be resolved no sooner than when the children are managed by the RubberTile widget, so that a client can use forward referencing of children.

XtNrefWidget

Class	Type	Default	Access
XtCRefWidget	XtPointer	NULL	SGI

Synopsis: The reference widget by widget ID.
Values: The ID of an existing child of the RubberTile.

If both `XtNrefName` and `XtNrefWidget` are given, they must agree.

XtNspace

Class	Type	Default	Access
XtCSpace	Dimension	0	SGI

Synopsis: The amount of space between the child and its reference widget (the widget identified by `XtNrefName` or `XtNrefWidget`).
Values: 0 ≤ `XtNspace`

Note – The `XtNspace` constraint resource conflicts with the Caption widget resource of the same name. Thus, setting this resource for a Caption widget that is a child of a RubberTile will affect both of the widgets.

XtNweight

Class	Type	Default	Access
XtCWeight	Dimension	1	SGI

Synopsis: The weight that dictates how much of a resize is applied to the child.

Values: 0 ≤ `XtNweight`

Activation Types

The following table lists the activation types used by the RubberTile.

Table 9-12 RubberTile Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

The RubberTile widget has no activation types besides the ones in “Common Activation Types” on page 68.

See Also

“Form Widget” on page 385.

9

Scrollbar Widget

Scrollbar Widget

Class

Class Name: `Scrollbar`
Class Pointer: `scrollbarWidgetClass`

Ancestry

Core-Primitive-Scrollbar

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/Scrollbar.h>
```

Description

The Scrollbar widget is similar to the slider widget, but provides additional features. It is typically used with a pane when the pane's contents exceed its size. The user can adjust the pane's view of the contents by manipulating the Scrollbar.

Components

Each full scrollbar has the following parts:

- Cable, indicating the extent of scrolling
- Anchors, top (left) and bottom (right), located at both ends of the cable. They are used to move the view to the corresponding extreme of the item or list of items being viewed.
- Elevator, which slides along the length of the cable, containing
  - Arrows, up (left) and down (right), used to move the view in the direction of the arrow by one unit.
  - Drag area, for moving the view by tracking the position of the mouse pointer relative to the scrollbar.

- Proportion indicator, which moves along with the elevator to indicate the size of the view and its position relative to the entire item or list of items being viewed.
- Page indicator (optional), located next to the drag area, which indicates the page number of the content being viewed. The page indicator will be displayed only when the SELECT button is pressed in the drag area.
- Scrollbar menu.

Because a scrollbar can be seen and used horizontally as well as vertically, the top anchor and bottom anchor have the aliases left anchor and right anchor, respectively.

Each scrollbar is associated with a content, as defined by the application. The content is composed of units (e.g. lines of text) that are visible in a viewing area. For a scrollbar to be useful, the content typically has more units than can fit in the viewing area. Hence, "scrolling" the content brings units into view as other units move out of view. The amount of the Content that is visible at one time is called a *pane* in the descriptions below.

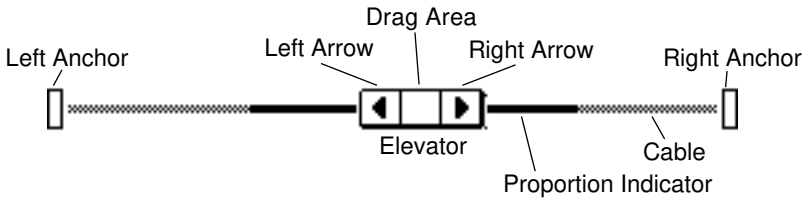

Figure 9-3 Scrollbar Horizontal Orientation

Figure 9-4

Subwidgets

The Scrollbar contains one subwidget: a MenuShell created automatically, and accessible through the `XtNmenuPane` resource.

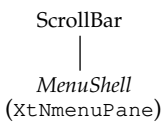

Figure 9-5 Scrollbar Subwidgets

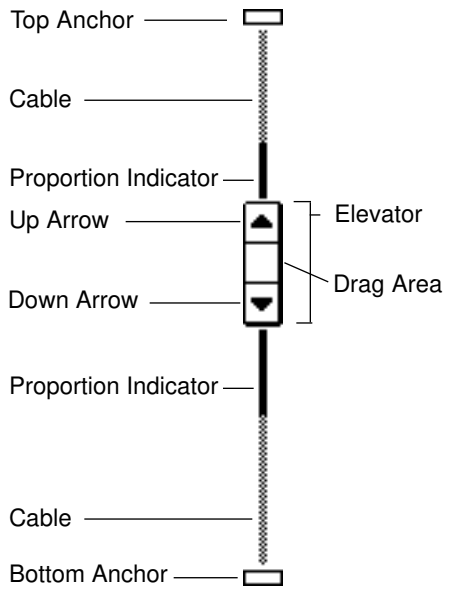

Figure 9-6 Scrollbar Vertical Orientation

Abbreviated Scrollbar

The Scrollbar widget responds to a parent's request to resize smaller by shortening the cable (and proportion indicator), but leaving the other elements full-sized. The Scrollbar widget will eliminate the cable entirely, if necessary, to meet a resize request. These abbreviated scrollbars are shown in the following figure. If necessary, the Scrollbar widget will eliminate the anchors (in addition to the cable and drag area) to meet a resize request to form a minimum Scrollbar.

Figure 9-7 Abbreviated Scrollbars

Elevator Motion

As visual feedback to the user, the elevator moves up and down (or left and right) along the line of the cable as the content scrolls or changes panes.

The range of motion of the elevator is not necessarily the full distance between the anchors. The application decides how far the elevator can be moved by evaluating each attempt to move it.

The user manipulates the scrollbar by pressing or clicking SELECT. The action performed depends on the position of the pointer and whether the application is willing to scroll the content.

Scrolling One Unit

Clicking SELECT on one of the arrows moves the elevator in the direction of the arrow, moves the pointer to stay on the arrow, and changes the content to move one unit out of view and another unit into view, such that the view scrolls in the opposite direction of the elevator motion.

If the application cannot scroll at this time, the elevator and pointer do not move and the view does not change.

Pressing SELECT on an arrow repeats the action described above. When SELECT is clicked or pressed, the arrow highlights while the scrolling action takes place. The highlighting stays until SELECT is released.

When the elevator has reached the end of the cable, the arrow in that direction is made inactive.

Scrolling Several Units

Dragging SELECT on the drag area moves the elevator along the cable to track the component of the pointer motion parallel to the cable. The content scrolls in the opposite direction, bringing one or more units into view as other units move out of view.

If granularity is enforced and the elevator is moved to a position that represents a non-integral number of units, the closest integral number of units is considered instead. If granularity is not enforced, the elevator is moved by the non-integral number of units. The `XtNsliderMoved` callback allows the application to enforce granularity.

When the application reaches the limit that it can scroll, the view no longer changes and the elevator stops moving. While dragging SELECT, the drag area highlights. The pointer is constrained to stay within the Drag Area as the elevator moves.

Scrolling Limits

Clicking SELECT on one of the anchors causes the view of the Content to change to the corresponding pane, and moves the elevator to the limit in the direction of the anchor. If the elevator is already at the limit, nothing happens.

Clicking SELECT on an anchor highlights the anchor while the scrolling action takes place.

Scrolling a Pane

Clicking SELECT on the cable above/left-of or below/right-of the elevator causes the view of the Content to change to the previous or next pane, respectively. The pointer is moved along the direction of the elevator travel to keep it off the elevator.

If only a partial pane remains before the limit of the Content is reached, the effect is as if the user clicked SELECT on the corresponding anchor. If the application cannot move to another pane, the view does not change, the elevator and pointer do not move.

Pressing SELECT on the Cable repeats the action described above.

Elevator at Limits

The application calibrates the scrollbar so that the position of the elevator on the scrollbar is in units useful to the application. In general, these units will not be pixels or points.

If the scrollbar is close enough to an anchor, the separation in application units may be zero pixels, because of the discrete nature of pixels. Here, the elevator is kept away from the anchor so that two points of the cable length are visible. The elevator is placed at the limit of motion only when the user explicitly moves the elevator to an anchor by clicking SELECT on the anchor, or drags the elevator until it reaches the limit.

Indicating View Proportion

The proportion indicator gives a gross measure of what part of the content is in view. Its size relative to the length of the cable is the same as the size of the pane relative to the size of the content. However, the scrollbar widget does not maintain this relation but relies on the application to provide the length of the proportion indicator.

The proportion indicator moves with the elevator such that both reach the limits together. When the content is scrolled to the beginning, the proportion indicator and the elevator align at the left or top end of the scrollbar. When the content is scrolled to the end, the proportion indicator and the elevator align at the right or bottom end of the scrollbar. For intermediate positions, the elevator is positioned proportionally between the ends of the proportion indicator. Thus, as the content is scrolled at a constant rate (e.g., by dragging SELECT), the elevator creeps from one end of the Proportion Indicator to the other at a constant rate.

Scrollbar Menu

The scrollbar menu (not shown in the figures) pops up when the user presses MENU anywhere over the Scrollbar widget. The menu has three default choices depending on the Scrollbar orientation.

`Here to top/left`	Scrolls the content so that the unit next to the pointer is placed at the top or left of the viewing area.
`Top/left to here`	Scrolls the content so that the unit at the top or left of the viewing area is placed next to the pointer.
`Previous`	Scrolls the content to restore the previous view. The Scrollbar widget remembers only the last two scroll positions, so repeated access of this choice alternates the content between two views.

An application can add choices to this menu, using the same technique for populating other menus (see “MenuShell Widget” on page 414). The ID of the Menu widget is available as a resource of the Scrollbar.

Coloration

For 3D and 2D, the area surrounding the Scrollbar is drawn with the parent's `XtNbackground`. `XtNforeground` is used to draw the optional page number indicator value.

For 3D, the Scrollbar coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1.

For 2D, `XtNbackground` and `XtNforeground` are used to render the Scrollbar as described by the *OPEN LOOK GUI Functional Specification*, Chapter 7, "Scrolling."

Keyboard Traversal

The Scrollbar's default value of the `XtNtraversalOn` resource is FALSE. If the application sets it to TRUE, undefined results occur.

Scrollbar Menu

The default choices in the Scrollbar Menu are created with `XtNtraversalOn` set to TRUE and `XtNmnemonic` set to the first character of their label.

When the Scrollbar menu is posted via keyboard traversal, the "Here to top" and "Top to here" buttons are not sensitive. These buttons depend on the position of the pointer when the menu is posted, and so they are not applicable when the menu is posted from the keyboard.

Keyboard Mnemonic Display

The Scrollbar does not display the mnemonic accelerator. If the Scrollbar is the child of a Caption widget, the Caption widget can be used to display the mnemonic for the Scrollbar.

Keyboard Accelerator Display

The Scrollbar does not display or respond to a keyboard accelerator because clicking the SELECT button on a Scrollbar activates depending on the pointer position.

Resources

Table 9-13 Scrollbar Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-14 Scrollbar Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	SGI
XtNreferenceName	String	NULL	GI

Table 9-14 Scrollbar Primitive Resources (Continued)

Name	Type	Default	Access
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	FALSE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 9-15 Scrollbar Resources

Name	Type	Default	Access
XtNcurrentPage	int	1	SGI
XtNdragCBType	OlDefine	OL_CONTINUOUS	SGI
XtNgranularity	int	1	SGI
XtNhereToLeftLabel	OlStr	"Here To Left"	GI
XtNhereToLeftMnemonic	unsigned char	'\0'	GI
XtNhereToTopLabel	OlStr	"Here To Top"	GI
XtNhereToTopMnemonic	unsigned char	'\0'	GI
XtNinitialDelay	int	500	SGI
XtNleftToHereLabel	OlStr	"Left To Here"	GI
XtNleftToHereMnemonic	unsigned char	'\0'	GI
XtNmenuPane	Widget	NULL	G
XtNmenuTitle	OlStr	"Scrollbar"	GI
XtNorientation	OlDefine	OL_VERTICAL	GI
XtNpointerWarping	Boolean	TRUE	SGI
XtNpreviousLabel	OlStr	"Previous"	GI
XtNpreviousMnemonic	unsigned char	'\0'	GI
XtNproportionLength	int	(`XtNsliderMax - XtNsliderMin`)	SGI
XtNrepeatRate	int	100	SGI
XtNshowPage	OlDefine	OL_NONE	SGI
XtNsliderMax	int	100	SGI
XtNsliderMin	int	0	SGI
XtNsliderMoved	XtCallbackList	NULL	SGIO
XtNsliderValue	int	0	SGI
XtNstopPosition	OlDefine	OL_ALL	SGI

Table 9-15 Scrollbar Resources (Continued)

Name	Type	Default	Access
XtNtopToHereLabel	OlStr	"Top To Here"	GI
XtNtopToHereMnemonic	unsigned char	'\0'	GI
XtNuseSetValCallback	Boolean	FALSE	SGI

The following table lists resources passed to the Menu subwidget maintained by the Scrollbar. They become resources of the Menu subwidget and can be set and read just like any other resources for the Scrollbar.

Table 9-16 Scrollbar Subwidget Resources[1]

Name	Type	Default	Access
XtNcenter	Boolean	TRUE	I
XtNhPad	Dimension	4	I
XtNhSpace	Dimension	4	I
XtNlayoutType	OlDefine	OL_FIXEDROWS	I
XtNmeasure	int	1	I
XtNpushpin	OlDefine	OL_NONE	I
XtNpushpinDefault	Boolean	FALSE	I
XtNsameSize	OlDefine	OL_COLUMNS	I
XtNvPad	Dimension	4	I
XtNvSpace	Dimension	4	I

1. These subwidget resources are described in the sections "ControlArea Widget" on page 249 and "MenuShell Widget" on page 414.

XtNcurrentPage

Class	Type	Default	Access
XtCCurrentPage	int	1	SGI

Synopsis: The page number displayed when the `XtNshowPage` resource is set to OL_RIGHT or OL_LEFT.

Values: 1 ≤ `XtNcurrentPage`

If `XtNshowPage` is set via `XtSetValues()`, or if `XtNshowPage` is already set and later `XtNsliderValue` is changed via `SetValues()`, it is the responsibility of the application to update `XtNcurrentPage` appropriately.

XtNdragCBType

Class	Type	Default	Access
XtCDragCBType	OlDefine	OL_CONTINUOUS	SGI

Synopsis: The frequency of issuing `XtNsliderMoved` callbacks during a drag operation.

Values: `OL_CONTINUOUS/"continuous"` – Issue callbacks continuously.
`OL_GRANULARITY/"granularity"` – Issue callbacks only when the drag box crosses any granularity positions.
`OL_RELEASE/"release"` – Issue callbacks only once when the SELECT button is released.

XtNgranularity

Class	Type	Default	Access
XtCGranularity	int	1	SGI

Synopsis: The distance the elevator attempts to move when clicking or pressing SELECT on an arrow.

Values: 1 ≤ `XtNgranularity` ≤ `XtNsliderMax` – `XtNsliderMin`

Normally, the drag operation does not honor granularity unless enforcement is set in the `XtNsliderMoved` callback procedure.

XtNhereToLeftLabel/ XtNhereToTopLabel/ XtNleftToHereLabel/ XtNpreviousLabel/ XtNtopToHereLabel

Class	Type	Default	Access
XtCHereToLeftLabel	OlStr	"Here to Left"	GI
XtCHereToTopLabel	OlStr	"Here to Top"	GI
XtCLeftToHereLabel	OlStr	"Left to Here"	GI
XtCPreviousLabel	OlStr	"Previous"	GI
XtCTopToHereLabel	OlStr	"Top to Here"	GI

Synopsis: The labels for the various menu buttons.

Values: Any `OlStr` values valid in the current locale.

XtNhereToLeftMnemonic/ XtNhereToTopMnemonic/ XtNleftToHereMnemonic/ XtNpreviousMnemonic/ XtNtopToHereMnemonic

Class	Type	Default	Access
XtCHereToLeftMnemonic	unsigned char	'\0'	GI
XtCHereToTopMnemonic	unsigned char	'\0'	GI
XtCLeftToHereLabel	unsigned char	'\0'	GI
XtCPreviousLabel	unsigned char	'\0'	GI
XtCTopToHereLabel	unsigned char	'\0'	GI

Synopsis: The mnemonics for the various menu buttons.
Values: Any ASCII character.

XtNinitialDelay

Class	Type	Default	Access
XtCInitialDelay	int	500	GI

Synopsis: The time, in milliseconds, before the first action occurs when SELECT is pressed on the cables or arrows.
Values: 0 < `XtNinitialDelay`

XtNmenuPane

Class	Type	Default	Access
XtCMenuPane	Widget	NULL	G

Synopsis: The widget where the scrollbar menu items are attached.
Values: The ID of the menu widget associated with the scrollbar.

This value is available once the Scrollbar has been created.

The default scrollbar menu contains the following items:

- Here To Top (Here To Left)
- Top To Here (Left To Here)
- Previous

The application must not remove these items from the menu.

The scrollbar menu can be enhanced by adding new menu items to this widget. Items can be added just as they are added to the menupane for a MenuShell or MenuButton widget.

XtNmenuTitle

Class	Type	Default	Access
XtCMenuTitle	OlStr	"Scrollbar"	GI

Synopsis: The title of the Scrollbar menu.
Values: Any `OlStr` value valid in the current locale.

XtNorientation

Class	Type	Default	Access
XtCOrientation	OlDefine	OL_VERTICAL	GI

Synopsis: The direction of the visual presentation of the widget.
Values: `OL_HORIZONTAL/"horizontal"` – Define a horizontal scrollbar.
`OL_VERTICAL/"vertical"` – Define a vertical scrollbar.

This resource cannot be changed via `XtSetValues()`.

XtNpointerWarping

Class	Type	Default	Access
XtCPointerWarping	Boolean	TRUE	SGI

Synopsis: Whether the pointer will move along with the elevator when SELECT is pressed on one of the arrows.
Values: `TRUE/"true"` – The pointer moves with the elevator.
`FALSE/"false"` – The pointer doesn't move with the elevator.

XtNproportionLength

Class	Type	Default	Access
XtCProportionLength	int	`(XtNsliderMax - XtNsliderMin)`	SGI

Synopsis: The size of the proportion indicator.
Values: 1 ≤ `XtNproportionLength` ≤ (`XtNsliderMax - XtNsliderMin`)

The application uses the `XtNsliderMax` and `XtNsliderMin` resources to calibrate the scrollbar, making its overall length correspond to the overall length of the content, and uses the `XtNproportionLength` resource to indicate how much of the content is visible. While this resource gives the

overall length of the proportion indicator, the elevator always covers part of it. If the elevator would completely hide the proportion indicator, 3-point sections of it are shown above and below (or left of and right of) the elevator. If the elevator is too close to an anchor to show all of a 3-point section, as much as possible of the section is shown on that side (this may be a zero-length section).

XtNrepeatRate

Class	Type	Default	Access
XtCRepeatRate	int	100	SGI

Synopsis: The time in milliseconds between repeated actions when SELECT is pressed on the cables or arrows.
Values: 0 ≤ `XtNrepeatRate`

XtNshowPage

Class	Type	Default	Access
XtCShowPage	OlDefine	OL_NONE	SGI

Synopsis: The position of the page indicator.
Values: `OL_LEFT/"left"` – The page indicator is to the left of the drag box.
`OL_RIGHT/"right"` – The page indicator is to the right of the drag box.
`OL_NONE/"none"` – The page indicator is not displayed.

The page indicator is displayed only if `XtNorientation` is OL_VERTICAL. The page indicator is popped up when the scrollbar is dragged by its drag area. While dragging, the page indicator is updated constantly. If `XtNshowPage` changes from OL_NONE to any other value, a popup window for the page indicator is created. If the value changes to OL_NONE, the popup window is destroyed.

XtNsliderMax/ XtNsliderMin

Class	Type	Default	Access
XtCSliderMax	int	100	SGI
XtCSliderMin	int	0	SGI

Synopsis: The calibration of the scrollbar.
Values: `XtNsliderMin < XtNsliderMax`

An application should set the values of these resources to correspond to the range of the content, and should set the value of the `XtNproportionLength` resource to the length of the view into the content. This calibrates the scrollbar.

The Scrollbar uses the calibration to convert the pixel location of the elevator into a value in the range represented by the length of the Content.

The explanation for this range relation follows: First, an application calibrates the scrollbar as described above, so that `XtNsliderMin` and `XtNsliderMax` span the length of the Content and `XtNproportionLength` gives the length of the view of the Content. That is,

```
XtNsliderMin          = start of Content
XtNsliderMax          = length of Content
XtNproportionLength   = length of Pane
```

Consider that the Elevator tracks a fixed position in the view; the position is arbitrary, but remains the same as the view is scrolled over the Content. This can be the first line in the view.

When the view is at the top of the content, the elevator is at the top of the scrollbar and the calibrated position of the first line is `XtNsliderMin`. However, when the view is at the bottom of the content, the elevator is at the bottom of the scrollbar and the calibrated position of the first line is `XtNsliderMax - XtNproportionLength`.

XtNsliderMoved

Class	Type	Default	Access
XtCSliderMoved	XtCallbackList	NULL	SGIO

Synopsis: The callback lists used when the scrollbar is manipulated.

The Scrollbar widget passes the final location of the Elevator, as an integer between `XtNsliderMin` and `XtNsliderMax` inclusive, in a structure pointed to by the *call_data* parameter:

```
typedef struct _OlScrollbarVerify {
    int         new_location;
    int         new_page;
    Boolean     ok;
    int         slidermin;
    int         slidermax;
```

```
    int         delta;
    Boolean     more_cb_pending;
} OlScrollbarVerify;
```

new_location — When the `XtNsliderMoved` callbacks are made, the *new_location* member gives the position of the attempted scroll. This will be the new value of the `XtNsliderValue` resource if the scroll attempt is successful; however, the `XtNsliderValue` resource is not updated until after the callbacks return.

new_page — Used to set the page number. To see the page number, the application must set `XtNshowPage` to OL_LEFT or OL_RIGHT.

ok — Initially set to TRUE; the application sets the value to indicate whether the scroll attempt is allowed. Since more than one callback routine may be registered for these resources, each callback routine can first check the *ok* member to see if a previous callback routine in the list has already rejected the scroll attempt. The Scrollbar will complete the scroll attempt only if, after the last callback has returned, the *ok* member is still TRUE.

If *ok* is FALSE after the last callback returns, the Scrollbar restores the Elevator to the position it was in before the user attempted to move it. This is required only when the Elevator has been dragged. The Scrollbar does not move the Elevator for other scrollbar manipulations until the scroll attempt has been verified.

slidermin — The same value as in the `XtNsliderMin` resource.

slidermax — The same value as in the `XtNsliderMax` resource.

delta — The distance between the new scroll position and the old, as a signed value.

more_cb_pending — Specifies if there are more callbacks pending.

A callback can change the *new_location* value to reflect a partial scroll. For example, if the scrolling granularity causes a scroll attempt past the end of an application's partially full buffer, the application should adjust *new_location* to a value representing the end of the buffer. The adjusted value must lie between the values present before the attempted scroll and the new values given in the `OlScrollbarVerify` structure.

The `XtNsliderMoved` callbacks are issued when the elevator position has been conditionally changed by the user

- clicking or pressing SELECT on the up/left or down/right arrow buttons;
- moving the Elevator to a new position by dragging SELECT on the drag area;
- clicking SELECT on the top/left or bottom/right anchors;
- clicking or pressing SELECT on the Cable.

XtNsliderValue

Class	Type	Default	Access
XtCSliderValue	int	0	SGI

Synopsis: The current position of the elevator.
Values: `XtNsliderMin` ≤ `XtNsliderValue` ≤ `XtNsliderMax` – `XtNproportionLength`

The Scrollbar widget keeps this resource up to date; however, an application can also get the current value through the `XtNsliderMoved` callbacks.

XtNstopPosition

Class	Type	Default	Access
XtCStopPosition	OlDefine	OL_ALL	SGI

Synopsis: The disposition of the drag box at the end of a drag operation.
Values: `OL_ALL/"all"` – Upon the release of the SELECT button in a drag operation, the drag box will be positioned at where it stops. `OL_GRANULARITY/"granularity"` – The drag box will snap to the nearest granularity position.

XtNuseSetValCallback

Class	Type	Default	Access
XtCUseSetValCallback	Boolean	FALSE	SGI

Synopsis: The callback action with programmatic control of slider value.
Values: `TRUE/"true"` – The callbacks specified in `XtNsliderMoved` are called when the slider value changes, even if it changes under program control instead of as the result of mouse action. `FALSE/"false"` – The callbacks specified in `XtNsliderMoved` are not called when the slider value is changed programmatically.

The callbacks listed in `XtNsliderMoved` are called as the result of mouse action by the user changing the slider value. This resource allows those callbacks to be called when the slider value changes under program control as well as by mouse action.

Activation Types

The following table lists the activation types used by the Scrollbar.

Table 9-17 Scrollbar Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_HSBMENU	ALT+CTRL R	XtNhorizSBMenuKey
OL_MENU	MENU	XtNmenuBtn
OL_MENUKEY	MENU	XtNmenuKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PAGEDOWN	ALT+PANE DOWN	XtNpageDownKey
OL_PAGELEFT	ALT+PANE LEFT	XtNpageLeftKey
OL_PAGERIGHT	ALT+PANE RIGHT	XtNpageRightKey
OL_PAGEUP	ALT+PANE UP	XtNpageUpKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_SCROLLBOTTOM	DATA END	XtNscrollBottomKey
OL_SCROLLDOWN	ALT+DOWN	XtNscrollDownKey
OL_SCROLLLEFT	ALT+LEFT	XtNscrollLeftKey
OL_SCROLLLEFTEDGE	ALT+{	XtNscrollLeftEdge
OL_SCROLLRIGHT	ALT+RIGHT	XtNscrollRightKey
OL_SCROLLRIGHTEDGE	ALT+}	XtNscrollRightEdge
OL_SCROLLTOP	DATA START	XtNscrollTopKey
OL_SCROLLUP	ALT+UP	XtNscrollUpKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey
OL_VSBMENU	CTRL R	XtNvertSBMenuKey

Activation types not described in the following table are described in "Common Activation Types" on page 68."

OL_MENU

This activation type pops up the Scrollbar menu as described in the *OPEN LOOK GUI Functional Specification* section "Scrollbar Menu" in Chapter 10. Activation of the Scrollbar menu items calls the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_MENUKEY/ OL_VSBMENU/ OL_HSBMENU

These activation types will pop up the Scrollbar menu. When the Scrollbar menu is posted via keyboard traversal, the "Here to top" and "Top to here" buttons will not be sensitive. These buttons will depend on the position of the pointer when the menu is posted, and so they will not be applicable when the menu is posted from the keyboard. The OL_VSBMENU activation type applies only to Scrollbars with an `XtNorientation` of OL_VERTICAL; and the OL_HSBMENU activation type applies only to Scrollbars with an `XtNorientation` of OL_HORIZONTAL.

OL_PAGEDOWN

For a scrollbar with `XtNorientation` of OL_VERTICAL, this activation type will decrement the slider value by `XtNproportionLength` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_PAGELEFT

For a scrollbar with `XtNorientation` of OL_HORIZONTAL, this activation type will decrement the slider value by the value of `XtNproportionLength` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_PAGERIGHT

For a scrollbar with `XtNorientation` of OL_HORIZONTAL, this activation type will increment the slider value by the value of `XtNproportionLength` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_PAGEUP

For a scrollbar with `XtNorientation` of OL_VERTICAL, this activation type will increment the slider value by `XtNproportionLength` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLBOTTOM

For a scrollbar with `XtNorientation` of OL_VERTICAL, this activation type will move the slider to the value of `XtNsliderMin` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLDOWN/ OL_SCROLLLEFT

These activation types will move the slider one negative unit of granularity and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLLEFTEDGE

For a scrollbar with `XtNorientation` of OL_HORIZONTAL, this activation type will move the slider to the value of `XtNsliderMin` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLRIGHTEDGE

For a scrollbar with `XtNorientation` of OL_HORIZONTAL, this activation type will move the slider to the value of `XtNsliderMax` minus the value of `XtNproportionLength` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLTOP

For a scrollbar with `XtNorientation` of OL_VERTICAL, this activation type will move the slider to the value of `XtNsliderMax` minus the value of `XtNproportionLength` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLUP/ OL_SCROLLRIGHT

These activation types will move the slider one positive unit of granularity and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SELECT

This activation type depends on the position of the pointer within the Scrollbar widget. When the pointer is positioned on the elevator, the `XtNsliderMoved` callback will be called according to the value of the `XtNdragCBType`. When the pointer is positioned on the right anchor, the behavior will be the same as the OL_SCROLLRIGHTEDGE activation type. When the pointer is positioned on the top anchor, the behavior will be the same as the OL_SCROLLTOP activation type. When the pointer is positioned on the left anchor, the behavior will be the same as the OL_SCROLLLEFTEDGE activation type. When the pointer is positioned on the bottom anchor, the behavior will be the same as the OL_SCROLLBOTTOM activation type. When the pointer is positioned on the up arrow, the behavior will be the same as the OL_SCROLLUP activation type. When the pointer is positioned on the right arrow, the behavior will be the same as the OL_SCROLLRIGHT activation type. When the pointer is positioned on the left arrow, the behavior will be the same as the OL_SCROLLLEFT activation type. When the pointer is positioned on the down arrow, the behavior will be the same as the OL_SCROLLDOWN activation type. When the pointer is positioned on the cable above or to the right of the elevator, the behavior will be the same as the OL_PAGEUP and OL_PAGERIGHT activation types, respectively. When the pointer is positioned on the cable below or to the left of the elevator, the behavior will be the same as the OL_PAGEDOWN and OL_PAGELEFT activation types, respectively.

See Also

"MenuShell Widget" on page 414.

ScrolledWindow Widget

Class

Class Name: `ScrolledWindow`
Class Pointer: `scrolledWindowWidgetClass`

Ancestry

Core-Composite-Constraint-Manager-ScrolledWindow

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/ScrolledWi.h>
```

Description

The ScrolledWindow has no native text or graphic capabilities, but provides the basis for implementing the OPEN LOOK scrollable text or graphics pane.

Components

The ScrolledWindow widget has the following components:

- Vertical scrollbar (typically)
- Horizontal scrollbar (typically)
- Content (not necessarily all visible)
- View of the content (visible part of content)
- View border

9

ScrolledWindow Widget

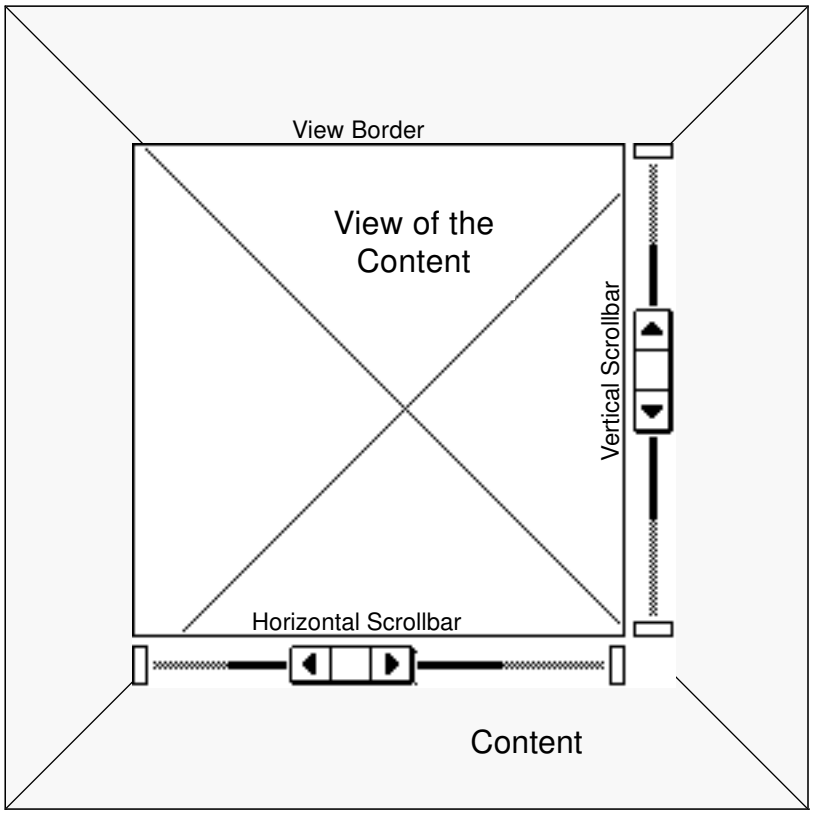

Figure 9-8 ScrolledWindow Components

Subwidgets

The ScrolledWindow contains five subwidgets: a BulletinBoard to hold the child contents; two Scrollbars, horizontal and vertical, along with their associated menus, created automatically, and accessible through the following resources:

- XtNhScrollbar
- XtNvScrollbar
- XtNhMenupane
- XtNvMenupane

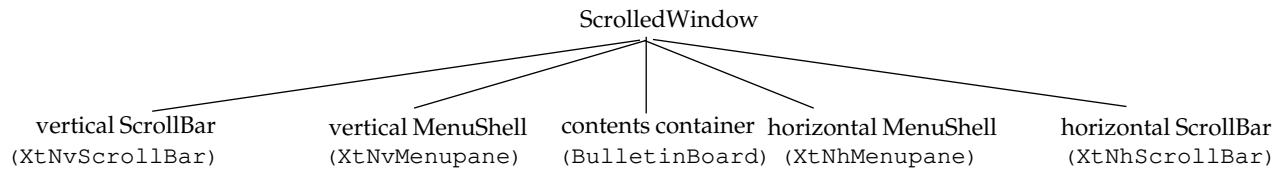

Figure 9-9 ScrolledWindow Subwidgets

View Border

The view border is a narrow outline around the view of the content: in 2D, it is a 1-pixel outline; in 3D, it is a 2-point beveled outline.

Content and View of Content

The ScrolledWindow widget implements a scrollable visible window (the *view of the content*) onto another, typically larger, data display (the *content*). The view can be moved through the content using the scroll bars.

To use the scrolled window, the application creates a widget capable of displaying the entire content as a child of the ScrolledWindow widget. The ScrolledWindow widget positions the child widget "within" the view of the content, and creates scroll bars for the horizontal and vertical dimensions, as needed. When the user performs some action on the scroll bars, the child widget will be repositioned accordingly within the view of the content. A larger child widget positioned within a smaller view by the ScrolledWindow widget, is forced to display only the viewed part of itself. This is all handled through normal widget geometry management.

The word "within" is used strictly in the widget sense: the larger child widget is positioned within the smaller view of the content part of the ScrolledWindow widget, which necessarily forces the child widget to display only the visible part of itself. The protocol for this is through normal widget geometry interactions.

Upper Left Corner Fixed on Resize

If the ScrolledWindow widget is resized, the upper left corner of the view stays fixed over the same spot in the content, unless this would cause the view to extend past the right or bottom edge of the content. If necessary, the upper left corner will shift left or up only enough to keep the view from extending past the right or bottom edge.

View Larger than Content

Generally, the view of the content never becomes larger than needed to show the content. This default behavior can be overridden using the `XtNrecomputeHeight` and `XtNrecomputeWidth` resources (see page 543).

Scrollbars

The scrollbars are configured to scroll integer values, in pixels, through the width and length of the content. This allows the finest degree of control of the positioning of the view of the content. However, the application can set the step rate through these values to avoid a large number of view updates as the user scrolls through the content.

Unless forced to appear (see "XtNforceHorizontalSB" on page 538), a scrollbar is removed from the side where it is no longer needed. Remaining scrollbars stay a fixed distance from the view.

Application Controlled Scrolling

The ScrolledWindow widget also supports application-controlled scrolling, including the scrolling of large amounts of data such as text. In this mode of operation, the application monitors user interaction with the Scrollbars and displays the appropriate data in the view.

The application specifies this mode of operation by setting the `XtNvAutoScroll/XtNhAutoScroll` resources to FALSE. Normally, these settings are combined with the setting of the `XtNvSliderMoved` and `XtNhSliderMoved` callbacks. Also, the application should specify an `XtNcomputeGeometries` callback, which is used to lay out the ScrolledWindow. To programmatically manipulate the scrollbars, the application must set the `XtNuseSetValCallback` resource of the Scrollbar widget to TRUE. The TextEdit widget (see page 623) recognizes when it is a child of a ScrolledWindow widget and operates in this mode.

Coloration

The diagram illustrates the resources that affect ScrolledWindow coloration.

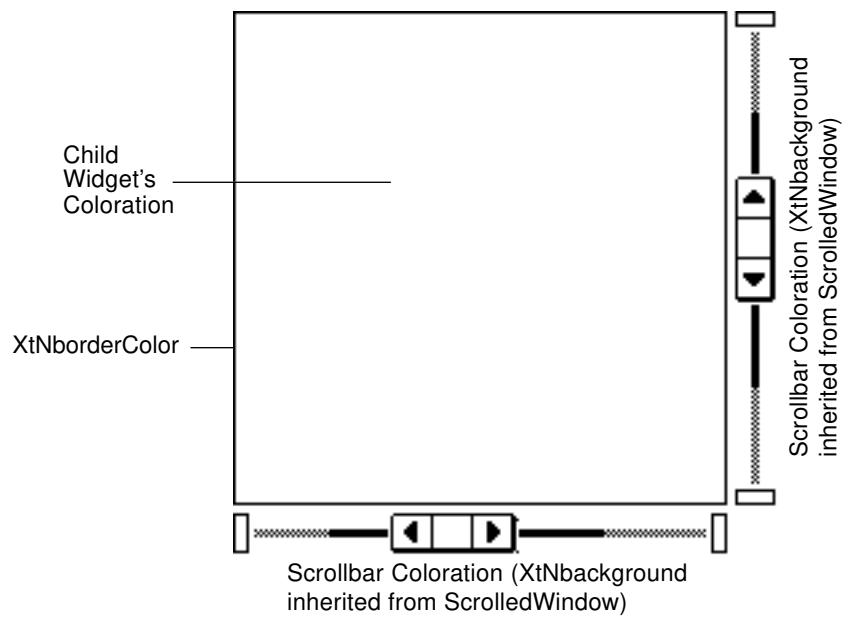

Figure 9-10 ScrolledWindow Coloration

Keyboard Traversal

The ScrolledWindow controls the keyboard traversal between the content, the horizontal ScrollBar, and the vertical ScrollBar. The scrollbars that are created by the ScrolledWindow have the `XtNtraversalOn` resource set to FALSE.

A content widget added to the ScrolledWindow with traversal enabled will be added to the traversable widgets in the window with the scrollbars so that the user can move between them with the NEXTFIELD (or MOVEUP or MOVELEFT) and PREVFIELD (or MOVEDOWN or MOVERIGHT) keys.

9

ScrolledWindow Widget Resources

Table 9-18 ScrolledWindow Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	(parent's)	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-19 ScrolledWindow Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 9-20 ScrolledWindow Manager Resources

Name	Type	Default	Access
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNinputFocusColor	Pixel	Red	SGID

Table 9-20 ScrolledWindow Manager Resources (Continued)

Name	Type	Default	Access
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNunrealizeCallback	XtCallbackList	NULL	SGIO
XtNuserData	XtPointer	NULL	SGI

Table 9-21 ScrolledWindow Resources

Name	Type	Default	Access
XtNalignHorizontal	OlDefine	OL_BOTTOM	SGI
XtNalignVertical	OlDefine	OL_RIGHT	SGI
XtNcomputeGeometries	Function	NULL	SGI
XtNcurrentPage	int	1	SGI
XtNforceHorizontalSB	Boolean	FALSE	SGI
XtNforceVerticalSB	Boolean	FALSE	SGI
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNhAutoScroll	Boolean	TRUE	SGI
XtNhInitialDelay	int	500 (msec)	SGI
XtNhMenuPane	Widget	NULL	G
XtNhRepeatRate	int	100 (msec)	SGI
XtNhScrollbar	Widget	NULL	G
XtNhSliderMoved	XtCallbackList	NULL	SGIO
XtNhStepSize	int	1	SGI
XtNinitialX	int	0	GI
XtNinitialY	int	0	GI
XtNrecomputeHeight	Boolean	TRUE	SGI
XtNrecomputeWidth	Boolean	TRUE	SGI
XtNshowPage	OlDefine	OL_NONE	SGI
XtNvAutoScroll	Boolean	TRUE	SGI
XtNviewHeight	Dimension	0	SGI
XtNviewWidth	Dimension	0	SGI
XtNvInitialDelay	int	500 (msec)	SGI
XtNvMenuPane	Widget	NULL	G
XtNvRepeatRate	int	100 (msec)	SGI

Table 9-21 ScrolledWindow Resources (Continued)

Name	Type	Default	Access
XtNvScrollbar	Widget	NULL	G
XtNvSliderMoved	XtCallbackList	NULL	SGIO
XtNvStepSize	int	1	SGI

XtNalignHorizontal

Class	Type	Default	Access
XtCAlignHorizontal	OlDefine	OL_BOTTOM	SGI

Synopsis: The placement of the horizontal scrollbar.
Values: `OL_BOTTOM/"bottom"` – Place the horizontal scrollbar on the bottom.
`OL_TOP/"top"` – Place the horizontal scrollbar on the top.

XtNalignVertical

Class	Type	Default	Access
XtCAlignVertical	OlDefine	OL_RIGHT	SGI

Synopsis: The placement of the vertical scrollbar.
Values: `OL_RIGHT/"right"` – Place the vertical scrollbar on the right.
`OL_LEFT/"left"` – Place the vertical scrollbar on the left.

XtNcomputeGeometries

Class	Type	Default	Access
XtCComputeGeometries	Function	NULL	SGI

Synopsis: A function called whenever the ScrolledWindow needs to lay out its children.

The content widget sets this resource to a pointer to a function that is called whenever the ScrolledWindow needs to lay out its children:

```
void (*compute_geometries)(
        Widget              content_widget,
        OlSWGeometries      *geometries);
```

The structure pointed to be *geometries* is defined as:

```
typedef struct _OlSWGeometries {
    Widget      sw;
    Widget      vsb;
    Widget      hsb;
    Dimension   bb_border_width;
    Dimension   vsb_width;
    Dimension   vsb_min_height;
    Dimension   hsb_height;
    Dimension   hsb_min_width;
    Dimension   sw_view_width;
    Dimension   sw_view_height;
    Dimension   bbc_width;
    Dimension   bbc_height;
    Dimension   bbc_real_width;
    Dimension   bbc_real_height;
    Boolean     force_hsb;
    Boolean     force_vsb;
} OlSWGeometries;
```

The ScrolledWindow widget populates the values in this structure before the call and examines them after the call to perform the layout operation. The fields are as follows:

sw	The widget ID of the ScrolledWindow widget.
vsb	The widget ID of its vertical Scrollbar widget.
hsb	The widget ID of its horizontal Scrollbar widget.
bb_border_width	The width of the border around the view.
vsb_width	The width of the vertical Scrollbar.
vsb_min_height	The minimum height of the vertical Scrollbar.
hsb_height	The height of the horizontal Scrollbar.
hsb_min_width	The minimum width of the horizontal Scrollbar.
sw_view_width	The width of the entire ScrolledWindow.
sw_view_height	The height of the entire ScrolledWindow.
bbc_width	The width of the view.
bbc_height	The height of the view.
bbc_real_width	The width of the contents.
bbc_real_height	The height of the contents.
force_hsb	When TRUE, a horizontal Scrollbar is forced to be present.
force_vsb	When TRUE, a vertical Scrollbar is forced to be present.

The called function is responsible for populating the following fields of this structure:

bbc_width
bbc_height
bbc_real_width
bbc_real_height
force_hsb
force_vsb

For example, if the application wants to make the view 300 pixels wide by 200 pixels high into a contents that is 1200 pixels wide and 700 pixels high, it needs to set these fields as follows:

bbc_width	=	300
bbc_height	=	200
bbc_real_width	=	1200
bbc_real_height	=	700

Since the view is smaller than the contents in both width and height, both scrollbars will appear in this case. Therefore, the values supplied for *force_vsb* and *force_hsb* are not important. However, one could have situations where one of the scrollbars is always wanted independent of the relationship between the view and content sizes; in such cases, *force_hsb* and *force_vsb* are useful.

XtNcurrentPage

Class	Type	Default	Access
XtCCurrentPage	int	1	SGI

Synopsis: The value to be used by the ScrolledWindow vertical scrollbar.

See “Scrollbar Widget” on page 508 for more details.

XtNforceHorizontalSB

Class	Type	Default	Access
XtCForceHorizontalSB	Boolean	FALSE	SGI

Synopsis: Force attachment of a horizontal scrollbar.
Values: `TRUE/"true"` – Disables size checking and forces the scrollbar to be attached to the window regardless of the size of the child widget.

FALSE/"false" – Performs size checking and does not force attachment.

When the child widget is created and positioned within the scrolled window, its width is examined. If the entire child widget will fit within the width of the scrolled window, and the horizontal scrollbar is not forced and the horizontal scrollbar will not be created, since there is no need to scroll in that direction.

If a scrollbar is forced but not needed because the content fits within the view, the scrollbar is made insensitive.

XtNforceVerticalSB

Class	Type	Default	Access
XtCForceVerticalSB	Boolean	FALSE	SGI

Synopsis: Force attachment of a vertical scrollbar.

Values: TRUE/"true" – Disables size checking and forces the scrollbar to be attached to the window, regardless of the size of the child widget.
FALSE/"false" – Performs size checking and does not force attachment.

When the child widget is created and positioned within the scrolled window, its height is examined. If the entire child widget will fit within the height of the scrolled window, and the vertical scrollbar is not forced, the vertical scrollbar will not be created, since there is no need to scroll in that direction.

If a scrollbar is forced but not needed because the content fits within the view, the scrollbar is made insensitive.

XtNforeground

Class	Type	Default	Access
XtCForeground	Pixel	XtDefaultForeground	SGID

Synopsis: The foreground color used by the ScrolledWindow to draw non-textual content.

Values: Any Pixel value valid for the current display, or any name from the $OPENWINHOME/lib/rgb.txt file. (Pixel values are used in C programs, rgb.txt values in X resource files. See "Resource Files" on page 7.)

XtNhAutoScroll/ XtNvAutoScroll

Class	Type	Default	Access
XtCHAutoScroll	Boolean	TRUE	SGI
XtCVAutoScroll	Boolean	TRUE	SGI

Synopsis: The scrolling mode in the horizontal (vertical) direction.

Values: `TRUE/"true"` – The ScrolledWindow widget is responsible for all interaction with the scrollbar and the positioning of the content window within the view.

`FALSE/"false"` – The application is responsible for all scrollbar interaction and scrolling of the data within the content window.

XtNhInitialDelay/ XtNvInitialDelay

Class	Type	Default	Access
XtCHInitialDelay	int	500 (msec)	SGI
XtCVInitialDelay	int	500 (msec)	SGI

Synopsis: The time in milliseconds of the initial repeat delay to be used when the scrolling arrows of the horizontal (vertical) scrollbar component of the ScrolledWindow are pressed.

XtNhMenuPane / XtNvMenuPane

Class	Type	Default	Access
XtCHMenuPane	Widget	NULL	G
XtCVMenuPane	Widget	NULL	G

Synopsis: A widget for a menu that can be popped up from the horizontal scrollbar.

These resources provide for the retrieval of a handle to the menu that may be popped up over a horizontal (vertical) scrollbar. This is useful for customizing the menu. For example, the application may add the concept of a page or chapter. The menu could then have items for scrolling forward and backward by those units.

See "Scrollbar Widget" on page 508 for more details.

XtNhRepeatRate/ XtNvRepeatRate

Class	Type	Default	Access
XtCHRepeatRate	int	100 (msec)	SGI
XtCVRepeatRate	int	100 (msec)	SGI

Synopsis: The time in milliseconds of the repeat delay to be used when the scrolling arrows of the horizontal (vertical) scrollbar component of the ScrolledWindow are pressed.

XtNhScrollbar/ XtNvScrollbar

Class	Type	Default	Access
XtCScrollbar	Widget	NULL	G
XtCScrollbar	Widget	NULL	G

Synopsis: The widget ID of the horizontal (vertical) scrollbar.

An application uses these values to set scrollbar characteristics, such as coloration.

XtNhSliderMoved/ XtNvSliderMoved

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback lists for tracking child position.

These resources mimic the `XtNsliderMoved` resource of the horizontal (vertical) scrollbar. The *call_data* parameter for this callback is a pointer to an `OlScrollBarVerify` structure, as in the Scrollbar widget. The application can validate a scroll attempt before the ScrolledWindow widget will reposition the view of the content, and can update the page number and adjust the scrollbar elevator position. See “Scrollbar Widget” on page 508 for more details.

XtNhStepSize/ XtNhStepSize

Class	Type	Default	Access
XtCHStepSize	int	1	SGI
XtCVStepSize	int	1	SGI

Synopsis: The minimum unit of horizontal (vertical) scrolling.
Values: The size in pixels of the minimum scrollable unit in the content:
0 < `XtNhStepSize`
0 < `XtNvStepSize`

For instance, to allow the user to scroll a single pixel in either direction, the value would be 1. Or, to allow the user to scroll a character at a time horizontally, the value would be the width of a character. Scrolling a character at a time requires a constant width font, of course.

The ScrolledWindow widget uses this value to calibrate the minimum scrolling step, `XtNgranularity`, of the scrollbars.

XtNinitialX/ XtNinitialY

Class	Type	Default	Access
XtCInitialX	int	0	GI
XtCInitialY	int	0	GI

Synopsis: The initial x- and y-position of the child widget.
Values: 0 ≥ `XtNinitialX`
0 ≥ `XtNinitialY`

The child widget is initially positioned at the upper left corner, (x,y) coordinates of (0,0). This positioning can be changed by specifying a new x,y location. The scrollbars are adjusted to give a visual indication of the offset specified in these resources. The content is positioned within the view of the content, so as the view of the content moves progressively further through the content, the coordinates of the position become more *negative*. Thus, the initial coordinate given in this resource should be zero or negative to ensure proper operation of the scrolled window.

XtNrecomputeHeight/ XtNrecomputeWidth

Class	Type	Default	Access
XtCRecomputeHeight	Boolean	TRUE	SGI
XtCRecomputeWidth	Boolean	TRUE	SGI

Synopsis: The resizing method used by the ScrolledWindow widget.

Values: `TRUE/"true"` – The ScrolledWindow shrinks the view of the content in the corresponding direction to absorb the change in the ScrolledWindow widget's size.
`FALSE/"false"` – The ScrolledWindow does not shrink the view in that direction.

These resources, together with `XtNviewWidth` and `XtNviewHeight`, are typically used to set a preferred dimension in a direction that should not be scrolled.

XtNshowPage

Class	Type	Default	Access
XtCShowPage	OlDefine	OL_NONE	SGI

Synopsis: The actions of the scrollbar page indicator during dragging. This value is directed to the vertical scrollbar in the ScrolledWindow widget.

Values: `OL_LEFT/"left"` – Display the page indicator to the left of the drag box.
`OL_RIGHT/"right"` – Display the page indicator to the right of the drag box.
`OL_NONE/"none"` – Display no page indicator.

See "Scrollbar Widget" on page 508 for more details.

Activation Types

The following table lists the activation types used by the ScrolledWindow.

Table 9-22 ScrolledWindow Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey

Table 9-22 ScrolledWindow Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_HELP	HELP	XtNhelpKey
OL_HSBMENU	HSBMENU	XtNhorizSBMenuKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PAGEDOWN	PAGEDOWN	XtNpageDownKey
OL_PAGELEFT	PAGELEFT	XtNpageLeftKey
OL_PAGERIGHT	PAGERIGHT	XtNpageRightKey
OL_PAGEUP	PAGEUP	XtNpageUpKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_SCROLLBOTTOM	SCROLLBOTTOM	XtNscrollBottomKey
OL_SCROLLDOWN	SCROLLDOWN	XtNscrollDownKey
OL_SCROLLLEFT	SCROLLLEFT	XtNscrollLeftKey
OL_SCROLLLEFTEDGE	SCROLLLEFTEDGE	XtNscrollLeftEdge
OL_SCROLLRIGHT	SCROLLRIGHT	XtNscrollRightKey
OL_SCROLLRIGHTEDGE	SCROLLRIGHTEDGE	XtNscrollRightEdge
OL_SCROLLTOP	SCROLLTOP	XtNscrolltopKey
OL_SCROLLUP	SCROLLUP	XtNscrollUpKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey
OL_VSBMENU	VSBMENU	XtNvertSBMenuKey

Activation types not described in the following list are described in "Common Activation Types" on page 68.

OL_HSBMENU

This activation type will pop up the menu associated with the `XtNhScrollbar` widget. When this menu is posted via keyboard traversal, the "Here to left" and "Left to here" buttons will not be sensitive. Activating the "Previous" menu item on this menu calls the `XtNhSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_PAGEDOWN

This activation type will decrement the `XtNvScrollbar`'s slider value by an amount necessary to scroll the view down by the height of the view, and call the `XtNvSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_PAGELEFT

This activation type will decrement the `XtNhScrollbar`'s slider value by an amount necessary to scroll the view left by the width of the view, and call the `XtNhSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_PAGERIGHT

This activation type will increment the `XtNhScrollbar`'s slider value by an amount necessary to scroll the view right by the width of the view, and call the `XtNhSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_PAGEUP

This activation type will increment the `XtNvScrollbar`'s slider value by an amount necessary to scroll the view up by the height of the view, and call the `XtNvSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLBOTTOM

This activation type will move the vertical scrollbar's slider such that the view is moved to the bottom of the content, and call the `XtNvSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLDOWN

This activation type will move the vertical scrollbar's slider one negative unit of granularity and call the `XtNvSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLLEFT

This activation type will move the horizontal scrollbar's slider one negative unit of granularity and call the `XtNhSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLLEFTEDGE

This activation type will move the horizontal scrollbar's slider such that the view is moved to the left edge of the content, and call the `XtNhSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLRIGHT

This activation type will move the horizontal scrollbar's slider one positive unit of granularity and call the `XtNhSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLRIGHTEDGE

This activation type will move the horizontal scrollbar's slider such that the view is moved to the right edge of the content, and call the `XtNhSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLTOP

This activation type will move the vertical scrollbar's slider such that the view is moved to the top of the content, and call the `XtNvSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLUP

This activation type will move the vertical scrollbar's slider one positive unit of granularity and call the `XtNvSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

OL_VSBMENU

This activation type will pop up the menu associated with the `XtNvScrollbar` widget. When this menu is posted via keyboard traversal, the "Here to top" and "Top to here" buttons will not be sensitive. Activating the "Previous" menu item on this menu calls the `XtNvSliderMoved` callback list with the appropriate `OlScrollbarVerify` structure.

See Also

"Scrollbar Widget" on page 508.

ScrollingList Widget

Class

Class Name: `ScrollingList`
Class Pointer: `scrollingListWidgetClass`

Ancestry

Core-Composite-Constraint-Manager-Form-ScrollingList

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/ScrollingL.h>
```

Description

The ScrollingList widget provides a list of items that the user can scroll through and choose. The application can make the choice of items exclusive, or allow the user to make multiple choices. It can also enable the user to edit items within the list.

OLIT releases after 3.1 have some new features and a new Application Programming Interface (API). The new features include multiple ScrollingList modes, item sensitivity, and double-click support. This *new API* enables the application to use these new features and also obsoletes the old API with a richer set of convenience functions. Refer to “ScrollingList Modes” on page 552 for more details.

Components

Each ScrollingList widget has the following parts: Items, a Scrollbar, and a View. If the application allows the list to be edited in place (in the view), the ScrollingList widget also uses an editable text field.

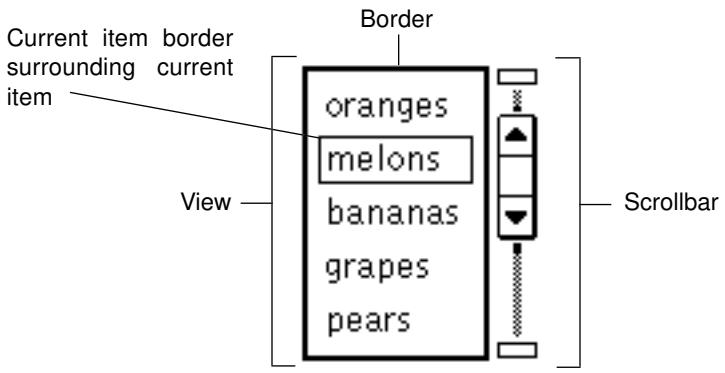

Figure 9-11 ScrollingList Components

Editable ScrollingList

The application can choose whether to allow the user to edit the items in a scrolling list. The editable text field is the interface for entering the new item, and is described later. Other aspects of the user interface for editing are controlled by the application. For example, the application can attach a menu to the scrolling list to allow the user to select where a new item is to be inserted, and can employ popup windows to gather additional information about a new item.

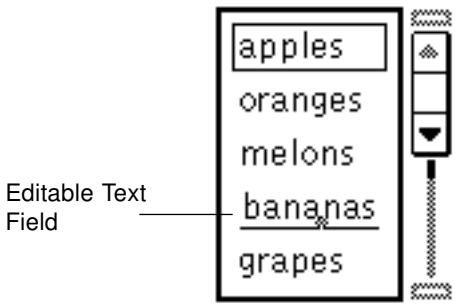

Figure 9-12 Editable ScrollingList

Subwidgets

The ScrollingList contains three subwidgets: a Scrollbar, a ListPane, and a TextField.

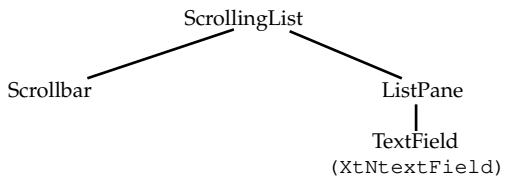

Figure 9-13 ScrollingList Subwidgets

Editable Text Field

The application can request that the ScrollingList widget provide an editable text field that will allow the user to change an existing item in the view. The TextField subwidget implements this editable text field. The ScrollingList widget manages the TextField widget as follows:

- Opening/closing – The application asks the ScrollingList widget to “open” and “close” the editable text field. Opening the editable text field widget maps it and positions it so that, as the user types in the name of a new or changed item, the name lines up with the existing item names. Closing the editable text field widget unmaps it. (As described below, there may be times when the widget is unmapped, yet still open.)
- Editing – If an existing item is being edited, the application requests the editable text field to overlay the item.
- Insertion – If a new item is being inserted, the application requests items to be scrolled down in the view to accommodate the editable text field.
- Mapping – The ScrollingList widget maps and unmaps the editable text field widget; the application does not.
- Scrolling – If the user scrolls the list while the editable text field is still open, the ScrollingList widget scrolls it with the rest of the items. If it has to be scrolled out of the view, it is scrolled out entirely, causing it to be unmapped but not closed. The application should not try to remap the child since it will be remapped when the list is scrolled back again.
- Setting/selecting – If the user attempts to make a selection or set a current item, the editable text field is automatically closed.

The application is responsible for handling the verification callbacks of the editable text field and for telling the ScrollingList widget to add a new item or change an existing item as a result of the user input.

Selectable ScrollingList

The application can choose whether to allow the user to select items from a scrolling list. If items can be selected, they can be copied elsewhere as text, and may be deletable ("cut").

Deleting Selected Items

The user can delete selected items. The ScrollingList widget provides some deletion capabilities through the selection mechanisms (see the discussion under "Text Selections on Items" on page 551), and the application can provide other capabilities, such as with a popup menu choice. The application verifies that each selected item can be deleted; it is responsible for providing feedback to the user for any items it will not delete. The ScrollingList widget updates the view to remove any deleted items.

Item Order

The list is assumed to have an order defined by the application. As it adds items, the application tells the ScrollingList widget where to insert them: either before an item already in the list or at the end of the list.

The application may change the content of a list at any time, including while it is displayed. The widget updates the view, if necessary, to reflect the changed list. To avoid unnecessary updates to the view when several changes need to be made, the application can instruct the ScrollingList widget to avoid updates until the changes are finished.

Making an Item Current

The user can make an item *current* by:

- Pressing SELECT over it, or
- Moving the input focus into the widget and typing the first letter of the item's name.

Since OLIT 3.2, either of these actions will add the item to the list of *current* items or will select a new current item, depending on the mode (exclusive,

exclusive-none-set, or nonexclusive). Double-clicking will invoke an application-specified callback.

In the default API, either of these actions caused a callback to the application, which could decide if the item should be made a current item, remain a current item, or be changed to a regular item, depending on the current state of the item and the needs of the application. Thus, the application can make the scrolling list behave as a set of exclusive or nonexclusive items.

Pressing SELECT also starts a selection, as described below.

Text Selections on Items

The ScrollingList widget allows selection operations on the items if the `XtNselectable` resource is TRUE; see page 567. Items that are moved or copied from the view are treated as a newline-separated list of text items, in the order they appear in the scrolling list, with no leading or trailing blanks on any item.

- Selecting a single item. Clicking SELECT on an item selects it and deselects any other active selection on the screen.
- Selecting other items. Clicking ADJUST on an item toggles its state, making an unselected item selected and a selected item unselected.
- Wipe-through selection, with SELECT. Pressing and dragging SELECT over items selects them and deselects any other active selection on the screen. The selection starts with the item where SELECT is pressed and extends to the item where SELECT is released. If the pointer moves above or below the view, the view scrolls additional items into the view, selecting them as well. The rate at which items scroll into the view is the same as when pressing SELECT on the up or down arrows of the Scrollbar. The pointer can move out of the view to the left or right without interrupting the selection.
- Wipe-through selection, with ADJUST. Pressing and dragging ADJUST marks the bounds of a selection the same way as pressing and dragging SELECT, except that the items covered are "toggled." (Previously selected items are deselected and previously unselected items are selected.)
- Copying items. Pressing COPY copies any selected items to the clipboard and deselects them.
- Cutting items. Pressing CUT moves any selected items to the clipboard and deletes them from the list. This operation is allowed only if the scrolling list is editable.

ScrollingList Modes

This feature is newly introduced in OLIT 3.2. The ScrollingList Widget can be created in one of the following modes:

OL_EXCLUSIVE	This "Exclusive" mode lets the user choose one item from a list of items, so that at any given time exactly one item is current or chosen.
OL_EXCLUSIVE_NONESET	This is a variation of the "Exclusive" ScrollingList. Here the user can choose one or none of the items on the list.
OL_NONEXCLUSIVE	This lets the user choose none, one, or multiple items from the list.
OL_NONE	None of the above three modes. This is the default mode for the ScrollingList and is termed the *default API* in this section. It is provided for backwards compatibility.

Note – The old API uses the `OlListToken` datatype; the new API uses the `OlSlistItemPtr` datatype instead. Applications cannot use any functions in the old API to manipulate the new features. Applications should also take care not to mix functions from the two APIs.

In this section the term *item* refers to a ScrollingList item of datatype `OlSlistItemPtr`.

Coloration

For both 2D and 3D, `XtNfontColor` is used to draw the ScrollingList item labels. If `XtNselectable` is set to TRUE, when a list item is selected (for COPY and PASTE), the ScrollingList reverses `XtNbackground` and `XtNfontColor` for the selected item. See individual "Coloration" sections for Scrollbar (page 514) and TextField (page 665) for information on the ScrollingList's scrollbar and textfield component coloration.

For 3D, ScrollingList item coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1.

For 2D, `XtNbackground` and `XtNforeground` are used to render the ScrollingList items as described by the *OPEN LOOK GUI Functional Specification*, Chapter 8, "Scrolling Lists."

If the toolkit resource `XtNmouseless` is set to TRUE and the toolkit resource `XtNinputFocusFeedback` is set to OL_INPUT_FOCUS_COLOR, then the background of the list item with focus will be drawn with the value of `XtNinputFocusColor` when the ScrollingList receives input focus. However, if `XtNinputFocusColor` is the same as `XtNbackground`, then the ScrollingList inverts `XtNfontColor` and `XtNbackground` within the list item with focus. Once the input focus leaves the widget, the original coloration is restored.

Keyboard Traversal

The default value of the `XtNtraversalOn` resource is TRUE.

The ScrollingList widget responds to the following keyboard navigation keys:

- NEXTFIELD moves to the next traversable widget in the window
- PREVFIELD moves to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application
- MOVEUP moves the input focus up one line
- MOVEDOWN moves the input focus down one line
- PANESTART moves the input focus to the first item in the pane
- PANEEND moves the input focus to the last item in the pane
- SCROLLUP scrolls up one item in the list
- SCROLLDOWN scrolls down one item in the list
- SCROLLTOP scrolls to the first item in the list
- SCROLLBOTTOM scrolls to the last item in the list
- PAGEUP scrolls up one page so that the first item visible is the last item visible in the pane
- PAGEDOWN scrolls down one page so that the last item visible is the first item visible in the pane

When an Editable Text Field is in the ScrollingList, the keyboard traversal keys defined for TextField widgets apply.

The SELECTKEY selects the Current Item and unselects any other active selection on the screen. The ADJUSTKEY toggles the Current Item's state, making an unselected Item selected and a selected Item unselected.

The scrolling keys of interest are defined within the ScrollingList and traversal to the Scrollbar is not necessary to manipulate the ScrollingList.

Keyboard Mnemonic Display

The ScrollingList widget displays the mnemonic accelerator for each item as part of its label. If the mnemonic character is in the label, then that character is displayed or highlighted according to the value of the toolkit resource `XtNshowMnemonics` (see page 16). If the mnemonic character is not in the label, it is displayed to the right of the label in parentheses and highlighted according to the value of `XtNshowMnemonics`.

If truncation is necessary, the mnemonic displayed in parentheses is truncated as a unit.

Known Deficiencies

ScrollingList does not support wide character string formats. The application should use the multibyte interface as a workaround.

Resources

Table 9-23 ScrollingList Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI

Table 9-23 ScrollingList Core Resources (Continued)

Name	Type	Default	Access
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-24 ScrollingList Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 9-25 ScrollingList Manager Resources

Name	Type	Default	Access
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNinputFocusColor	Pixel	Red	SGID
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNunrealizeCallback	XtCallbackList	NULL	SGIO
XtNuserData	XtPointer	NULL	SGI

Table 9-26 ScrollingList Resources

Name	Type	Default	Access
XtNalign	Boolean	TRUE	SGI
XtNapplAddItem[1]	OlApplAddItemProc	(special)	G
XtNapplDeleteItem[1]	OlApplDeleteItemProc	(special)	G
XtNapplEditClose[1]	OlApplEditCloseProc	(special)	G

Table 9-26 ScrollingList Resources (Continued)

Name	Type	Default	Access
XtNapplEditOpen[1]	OlApplEditOpenProc	(special)	G
XtNapplTouchItem[1]	OlApplItemProc	(special)	G
XtNapplUpdateView[1]	OlApplUpdateViewProc	(special)	G
XtNapplViewItem[1]	OlApplItemProc	(special)	G
XtNcurrentItems	OlSlistItemPtr *	NULL	G
XtNfirstViewableItem	OlSlistItemPtr	(calculated)	SG
XtNitemCurrentCallback	XtCallbackList	NULL	SGIO
XtNitemHeight	Dimension	0	SGI
XtNitemNotCurrentCallback	XtCallbackList	NULL	SGIO
XtNlastViewableItem	OlSlistItemPtr	(calculated)	SG
XtNlistPane	Widget	(special)	G
XtNmultiClickCallback	XtCallbackList	NULL	SGIO
XtNnumCurrentItems	int	0	G
XtNnumItems	int	0	G
XtNposition	OlDefine	OL_LEFT	SGI
XtNprefMaxWidth	Dimension	0	SGI
XtNprefMinWidth	Dimension	0	SGI
XtNrecomputeWidth	Boolean	TRUE	SGI
XtNscrollingListItems	OlSlistItemPtr *	NULL	G
XtNscrollingListMode	OlDefine	OL_NONE	GI
XtNselectable	Boolean	FALSE	SGI
XtNsensitive	Boolean	TRUE	GIO
XtNspace	Dimension	4	SGI
XtNtextField	Widget	(special)	G
XtNuserDeleteItems	XtCallbackList	NULL	SGIO
XtNuserMakeCurrent[1]	XtCallbackList	NULL	SGIO
XtNviewableItems	OlSlistItemPtr *	NULL	G
XtNviewHeight	Dimension	(calculated)	SGI

1. This resource cannot be used with the new API and related functions.

XtNalign

Class	Type	Default	Access
XtCAlign	Boolean	TRUE	SGI

Synopsis: The alignment of combined string/glyph items.
Values: `TRUE/"true"` – The second components of all items having both string and glyph are aligned, at an offset of `XtNspace` from the widest first component.
`FALSE/"false"` – The items are not aligned.

XtNapplAddItem

Class	Type	Default	Access
XtCApplAddItem	OlApplAddItemProc	(special)	G

Synopsis: The system-provided routine the application should use when it adds a new item to the list. Applications should use `XtGetValues()` to retrieve this routine.

Note – This resource cannot be used with the new API and related functions.

The prototype for this routine is as follows:

```
typedef OlListToken (*OlApplAddItemProc)(Widget widget,
                                         OlListToken parent,
                                         OlListToken reference,
                                         OlListItem item);
```

widget Identifies the ScrollingList widget instance.

parent Should be set to 0, for compatibility with future changes.

reference Identifies an item before which to insert the new item. This value can be zero to append the new item to the list.

item Describes the new item.

The content of the `OlListItem` structure is copied by the ScrollingList widget into space that it maintains; however, the data pointed to by the label and glyph members are not copied. The application can access the copied data directly, using the `OlListItemPointer()` macro to get a pointer to the `OlListItem` structure for the item.

If it changes the data, the application should use the `XtNapplTouchItem` routine to let the ScrollingList widget know the data has changed.

If mapped and if allowed by the application (see "XtNapplUpdateView" on page 561), the ScrollingList widget updates the view if the new item will be in the view. The view is changed as little as possible: if the new item is in the upper half of the view, the items above it are scrolled up and the top item is scrolled off; if the new item is in the lower half of the view, the items below it are scrolled down and the bottom item is scrolled off.

This routine is also used to build the item list from scratch.

XtNapplDeleteItem

Class	Type	Default	Access
XtCApplDeleteItem	OlApplDeleteItemProc	(special)	G

Synopsis: The system-provided routine the application should call when it deletes an item from the list. Applications should use `XtGetValues()` to retrieve this routine.

Note – This resource cannot be used with the new API and related functions.

The prototype for this routine is as follows:

```
typedef void (*OlApplDeleteItemProc)(Widget widget,
                                     OlListToken token);
```

widget Identifies the ScrollingList widget instance.

token Identifies the deleted item.

If mapped and if allowed by the application (see "XtNapplUpdateView" on page 561), the ScrollingList widget updates the view if the deleted item was visible.

The view is changed as little as possible:

- If the deleted item was in the upper half of the view, items above it are scrolled down and an item is scrolled in from the top;
- If the deleted item was in the lower half of the view, items below it are scrolled up and an item is scrolled in from the bottom.
- If the view is already at the top or bottom, the additional item is scrolled in from the other end, if possible.

XtNapplEditClose

Class	Type	Default	Access
XtCApplEditClose	OlApplEditClose	(special)	G

Synopsis: The system-provided routine the application should call when the user has finished editing an item in the view. Applications should use `XtGetValues()` to retrieve this routine.

Note – This resource cannot be used with the new API and related functions.

The prototype for this routine is as follows:

```
typedef void (*OlApplEditCloseProc)(Widget widget);
```

widget Identifies the ScrollingList widget instance.

When this routine is called, the ScrollingList widget unmaps the editable text field widget, scrolling up the items below it if they had been scrolled down to allow an insert.

The application is responsible for calling the `XtNapplAddItem` routine to add the new item, or calling the `XtNapplTouchItem` routine to mark the item as changed.

To avoid unnecessary updates to the view, the application should add the new item (`XtNapplAddItem`) or mark the changed item (`XtNapplTouchItem`) before closing the editable text field. A later call to the `XtNapplEditClose` routine without an intervening call to the `XtNapplEditOpen` routine is ignored.

If mapped, the ScrollingList widget updates the view, even if the application had halted updates (see "XtNapplUpdateView" on page 561). If the application had halted updates, they will continue to be halted afterwards.

XtNapplEditOpen

Class	Type	Default	Access
XtCApplEditOpen	OlApplEditOpen	(special)	G

Synopsis: The system-provided routine the application should use when it wants to allow the user to insert a new item or change an existing item in the view. Applications should use `XtGetValues()` to retrieve this routine.

Note – This resource cannot be used with the new API and related functions.

The prototype for this routine is as follows:

```
typedef void (*OlApplEditOpenProc) (Widget widget,
                                    Boolean insert,
                                    OlListToken reference);
```

widget Identifies the ScrollingList widget instance.

insert Tells whether items should be scrolled down to make room for inserting a new item. A value of FALSE implies that an item is being edited in place and no items are to be scrolled.

reference Identifies an item before which a new item is to be inserted (*insert* is TRUE) or identifies the item that is being changed (*insert* is FALSE). If *insert* is TRUE, this value can be zero to append a new item at the end of the list. If *insert* is FALSE, this value must refer to an existing item. The referenced item does not have to be in the view (see below).

If a new item is being inserted, the ScrollingList widget makes room for the editable text field by scrolling down the referenced item and any items below it. If the referenced item is not in the view, it is automatically made visible just as if the application had called the `XtNapplViewItem` routine first. The `XtNapplEditOpen` routine can be called again before an intervening call to the `XtNapplEditClose` routine. The effect is as if the `XtNapplEditClose` routine was called, but without multiple updates to the view. For example, this allows the application to let the user insert several new items in succession: the editable text field moves down as each item is inserted, but is never removed from the view.

Note – This item-insertion feature is currently not implemented.

If mapped, the ScrollingList widget updates the view, even if the application had halted updates (see "XtNapplUpdateView" on page 561). If the application had halted updates, they will continue to be halted afterwards.

XtNapplTouchItem

Class	Type	Default	Access
XtCApplTouchItem	OlApplTouchItemProc	(special)	G

Synopsis: The system-provided routine the application should use when it changes an item in the list. Applications should use `XtGetValues()` to retrieve this routine.

Note – This resource cannot be used with the new API and related functions.

The prototype for this routine is as follows:

```
typedef void (*OlApplTouchItemProc)(Widget widget,
                                    OlListToken token);
```

widget Identifies the ScrollingList widget instance.

token Identifies the item that has changed.

If mapped and if allowed by the application (see `XtNapplUpdateView`), the ScrollingList widget updates the view if the changed item is visible.

XtNapplUpdateView

Class	Type	Default	Access
XtCApplUpdateView	OlApplUpdateView	(special)	G

Synopsis: The system-provided routine the application can call to keep the ScrollingList widget from updating the view, or to let it again update the view. Applications should use `XtGetValues()` to retrieve this routine.

Note – This resource cannot be used with the new API and related functions.

The prototype for this routine is as follows:

```
typedef void (*OlApplUpdateViewProc)(Widget widget,
                                     Boolean ok);
```

widget Identifies the ScrollingList widget instance.

ok Either TRUE or FALSE, depending on whether the ScrollingList can update the View as it changes, or not, respectively.

From the time the `XtNapplUpdateView` routine is called with a FALSE argument until it is called with a TRUE argument, the ScrollingList does not

update the view in response to application-made changes. The ScrollingList widget updates the view once for each of these exceptions, each time an exception occurs. An application should use this routine to bracket a set of changes to avoid spurious changes to the view. This routine is not needed if only one change is made to the list. The following example illustrates the use of the `XtNapplUpdateView` routine.

```
    /* Stop view updates. */
(*applUpdateView)(widget, FALSE);
    /* Make some changes. */
(*applDeleteItem)(widget,...);
(*applDeleteItem)(widget,...);
(*applAddItem)(widget,...);
(*applTouchItem)(widget, ...);
    /* Allow the view to be updated again. */
(*applUpdateView)(widget, TRUE);
```

XtNapplViewItem

Class	Type	Default	Access
XtCApplViewItem	OlApplViewItemProc	(special)	G

Synopsis: The system-provided routine the application can call when it wants a particular item placed in the view. Applications should use `XtGetValues()` to retrieve this routine.

Note – This resource cannot be used with the new API and related functions.

The prototype for this routine is as follows:

```
typedef void (*OlApplViewItemProc)(Widget widget,
                                   OlListToken token);
```

widget Identifies the ScrollingList widget instance.

token Identifies the item to move into the view.

The item is moved into the view in a way that minimizes the change to the view. If the item is currently in the view, nothing is changed. If scrolling the list up or down brings the item into the view while keeping at least one previously viewed item in the view, the list is scrolled. Otherwise, the item is placed at the top of the view, or as close to the top as possible if there aren't enough items in the current Level to fill the view below it. If mapped and if allowed by the application (see "XtNapplUpdateView" on page 561), the ScrollingList widget updates the view. See also "Known Deficiencies" on page 585.

XtNcurrentItems

Class	Type	Default	Access
XtCCurrentItems	OlSlistItemPtr *	NULL	G

Synopsis: The array of `OlSlistItemPtr` items that are current in the list.

`XtGetValues()` for this resource returns the items that are current and the application must not free these items.

XtNfirstViewableItem

Class	Type	Default	Access
XtCFirstViewableItem	OlSlistItemPtr	(calculated)	SG

Synopsis: Which item is the first viewable item in the list. The default is the first item added to the list.

XtNitemCurrentCallback

Class	Type	Default	Access
XtCItemCurrentCallback	XtCallbackList	NULL	SGIO

Synopsis: The list of callbacks to be invoked when an item is made current by the user pressing the SELECT mouse button over an item.

This callback list is not invoked if `XtNscrollingListMode` is OL_NONE. The *call_data* structure is `OlSlistCallbackStruct` (see page 571) and the reason field in the *call_data* structure will be set to OL_REASON_ITEM_CURRENT.

XtNitemHeight

Class	Type	Default	Access
XtCItemHeight	Dimension	0	SGI

Synopsis: The height of each item.
Values: 0 ≤ `XtNitemHeight`

If set to zero, the height of each item will be the height of the font specified by `XtNfont`. If the items in the ScrollingList include glyphs, this resource should be set to the maximum height of all the glyphs.

XtNitemNotCurrentCallback

Class	Type	Default	Access
XtCItemNotCurrentCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the user presses the SELECT mouse button on a current item.

This callback list is not invoked if `XtNscrollingListMode` is OL_NONE. The *call_data* structure is `OlSlistCallbackStruct` (see page 571) and the *reason* field in the *call_data* structure will be set to OL_REASON_ITEM_NOT_CURRENT.

XtNlastViewableItem

Class	Type	Default	Access
XtCLastViewableItem	OlSlistItemPtr	(calculated)	SG

Synopsis: The last viewable item in the list.

XtNlistPane

Class	Type	Default	Access
XtCReadOnly	Widget	(special)	G

Synopsis: The ListPane child of the ScrollingList.

The ListPane widget is created as a child by the ScrollingList widget during instantiation.

The ListPane widget is subclassed off the Primitive widget. The ScrollingList items are rendered on the ListPane widget. `XtNconsumeEvent` callbacks or event handlers for the ScrollingList must be registered on this widget.

XtNmultiClickCallback

Class	Type	Default	Access
XtCMultiClickCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the user presses the SELECT mouse button on an item multiple times.

This callback list is not invoked if `XtNscrollingListMode` is OL_NONE. The *call_data* structure is `OlSlistCallbackStruct` (see page 571) and the reason field in the *call_data* structure will be set to OL_REASON_DOUBLE_CLICK. The `XtNmultiClickTimeout` toolkit resource defines the timer interval in

milliseconds within which two successive button clicks are interpreted as a multiclick.

XtNnumCurrentItems

Class	Type	Default	Access
XtCNumCurrentItems	int	0	G

Synopsis: The number of current items.

XtNnumItems

Class	Type	Default	Access
XtCNumItems	int	0	G

Synopsis: The total number of items in the list at any given time.

This resource automatically gets updated by the ScrollingList whenever an item gets added or deleted.

XtNposition

Class	Type	Default	Access
XtCPosition	OlDefine	OL_LEFT	SGI

Synopsis: The position of the string with respect to the glyph in items having both components.

Values: `OL_LEFT/"left"` – Sets the string to the left of the glyph.
`OL_RIGHT/"right"` – Sets the string to the right of the glyph.

XtNprefMaxWidth/ XtNprefMinWidth

Class	Type	Default	Access
XtCPrefMaxWidth	Dimension	0	SGI
XtCPrefMinWidth	Dimension	0	SGI

Synopsis: Controls dynamic resizing.

Values: $0 \leq$ `XtNprefMaxWidth`
$0 \leq$ `XtNprefMinWidth`

If the value of these resources is specified as zero, the static sizing behavior is preserved (which is the default). However, if these are specified as nonzero, then the width of the ScrollingList will be no less than `XtNprefMinWidth` and no greater than `XtNprefMaxWidth`, regardless of the length of the items

inside. Items that are longer than `XtNprefMaxWidth` will automatically be marked with a caret, indicating that they are not completely visible.

If the application wishes to set the List to a single width, then `XtNprefMinWidth` should equal `XtNprefMaxWidth`, thus forcing the ScrollingList width to a particular size.

XtNrecomputeWidth

Class	Type	Default	Access
XtCRecomputeWidth	Boolean	TRUE	SGI

Synopsis: Whether the ScrollingList widget should dynamically resize itself in the horizontal direction.

Values: `TRUE/"true"` – The ScrollingList grows horizontally to accommodate the widest item in the list.
`FALSE/"false"` – The ScrollingList does not grow horizontally to accommodate changes in item dimensions.

The ScrollingList never shrinks horizontally to accommodate changes in item dimensions.

XtNscrollingListItems

Class	Type	Default	Access
XtCScrollingListItems	OlSlistItemPtr *	NULL	G

Synopsis: The entire list of items in the widget.

The number of items in this list is determined by the resource `XtNnumItems`. If the widget is empty it returns NULL. `XtGetValues()` on this resource returns the items themselves and not a copy of the list items. The application must not free the returned items. The application must not free the returned items and must make a copy if this returned list has to be used in any of the convenience functions.

XtNscrollingListMode

Class	Type	Default	Access
XtCScrollingListMode	OlDefine	OL_NONE	GI

Synopsis: The mode of the ScrollingList.

Values: `OL_EXCLUSIVE` – Exclusive mode; one item is current at all times.
`OL_EXCLUSIVE_NONESET` – Exclusive-none-set mode; zero or

one item is current at all times.
`OL_NONEXCLUSIVE` – Nonexclusive mode; zero, one, or many items are current.
`OL_NONE` – Default API mode, for backwards compatibility.

XtNselectable

Class	Type	Default	Access
XtCSelectable	Boolean	FALSE	SGI

Synopsis: Whether the user can select items in the scrolling list.
Values: `TRUE/"true"` – Items can be selected with SELECT and ADJUST and copied with the COPY key. Items may be deleted with the CUT key, although the application can stop some or all selected items from being deleted.
`FALSE/"false"` – Items cannot be selected and the COPY and CUT keys have no effect.

XtNspace

Class	Type	Default	Access
XtCSpace	Dimension	4	SGI

Synopsis: The spacing between the string and the glyph in an item.
Values: 0 ≤ `XtNspace`

XtNtextField

Class	Type	Default	Access
XtCTextField	Widget	(special)	G

Synopsis: The widget ID of the editable text field widget.

This value is available once the ScrollingList widget has been created. The ScrollingList widget resets the following TextField resources before returning from each invocation of the `XtNapplEditOpen` routine:

Name	Class	Value
XtNwidth	XtCWidth	Width available in View
XtNstring	XtCString	Name of Item to be changed

XtNuserDeleteItems

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The list of callbacks invoked when the user tries to delete items from the list.

Currently, the only way the ScrollingList widget handles deletions is through a *cut* operation. The *call_data* parameter has different values depending on the mode of the ScrollingList.

In the default API (`XtNscrollingListMode` = OL_NONE), the *call_data* parameter points to a structure `OlListDelete` defined as:

```
typedef struct _OlListDelete {
    OlListToken    *tokens;
    Cardinal       num_tokens;
} OlListDelete;
```

tokens A list identifying the items to be deleted. The application is expected to act on each item separately, calling the `XtNapplDeleteItem` routine to delete each from the list. The application may refuse to delete some or all of the items, and is responsible for providing any feedback to the user.

num_tokens The number of items to delete.

In the new API (`XtNscrollingListMode` ≠ OL_NONE), the *call_data* parameter points to a structure `OlSlistUserDeleteCallbackStruct` defined as:

```
typedef struct {
    int                reason;
    XEvent             *event;
    OlDefine           mode;
    OlSlistItemPtr     *user_del_items;
    int                num_del_items;
} OlSlistUserDeleteCallbackStruct;
```

reason Indicates the reason for the callback.

mode Indicates what mode the scrollinglist is in:

OL_EXCLUSIVE
OL_EXCLUSIVES_NONESET
OL_NONEXCLUSIVE

event	Points to the `XEvent` that triggered the callback. This member can be NULL.
item	The item that was most recently made *current* when the event that triggered that callback occurred.
item_pos	Specifies the `Position` of the item in the list.
cur_items	Specifies a list of current items at the time of the event that triggered the callback. When the *mode* is OL_NONEXCLUSIVE, this gives a list of all the current items. When the *mode* is OL_EXCLUSIVE or OL_EXCLUSIVE_NONESET, this variable has the same value as the field *item*. This points to temporary storage that the application must copy into its own space if needed.
num_cur_items	Gives the number of current items in the list.
cur_items_pos	An array of integers, representing one for each current item. This points to temporary storage that the application must copy into its own space if needed.

and the *reason* field will be set to OL_REASON_USER_DELETE_ITEMS. It is up to the application to act on each item by calling `OlSlistDeleteItem()` or `OlSlistDeleteItems()` (see page 579) and providing necessary feedback.

XtNuserMakeCurrent

Class	Class	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the user presses SELECT over an item.

Note – This resource cannot be used with the new API and related functions.

The *call_data* parameter is the `OlListToken` value that identifies the item. The application is expected to decide if the current item status of this item should change. The *attributes* member of the `OlListItem` structure for this item is not automatically changed by the ScrollingList widget.

XtNviewableItems

Class	Type	Default	Access
XtCViewableItems	OlSlistItemPtr *	NULL	G

Synopsis: The list of viewable items in the widget.

The number of items in this list is determined by the `XtNviewHeight` resource. There are three possible cases: if `XtNnumItems` is zero the value returned is NULL. If the value of `XtNnumItems` is greater than zero but less than the value of `XtNviewHeight` then the list item pointer returned will be valid only for the first `XtNnumItems` in the list. Otherwise the list item pointer is valid for all `XtNviewHeight` items.
`XtGetValues()` on this resource returns the items themselves and not a copy of the list items. The application must not free the returned items. The application must make a copy if this returned list has to be used in any of the convenience functions.

XtNviewHeight

Class	Type	Default	Access
XtCViewHeight	Dimension	(calculated)	SGI

Synopsis: The preferred height of the view as the number of items to show.
Values: 0 ≤ `XtNviewHeight`

If a nonzero value is given, the corresponding `XtNheight` resource is computed by converting this number to pixels and adding any padding or border thickness. In this case, any value given in the `XtNheight` resource is overwritten. If a zero value is given in the `XtNviewHeight` resource, the `XtNheight` resource is used as an estimate. The view is sized to show an integral number of items, such that the overall height of the ScrollingList widget is less than or equal to `XtNheight`, if possible. However, the view is always large enough to show at least one item, and is no shorter than the minimum scrollbar size. If neither the `XtNviewHeight` resource nor the `XtNheight` resource is set, or both are set to zero, the view is made as small as possible, limited as described above.

Callback Information

OlSlistCallbackStruct

This structure is used by a number of the callbacks defined for the new API.

```
typedef struct {
    int                 reason;
    XEvent              *event;
    OlDefine            mode;
    OlSlistItemPtr      item;
    int                 item_pos;
    OlSlistItemPtr      *cur_items;
    int                 num_cur_items;
    int                 *cur_items_pos;
} OlSlistCallbackStruct;
```

reason	Indicates the reason for the callback.
mode	The mode of the scrollinglist: OL_EXCLUSIVE OL_EXCLUSIVES_NONESET OL_NONEXCLUSIVE
event	Points to the `XEvent` that triggered the callback. This member can be NULL.
item	The item that was most recently made *current* when the event that triggered that callback occurred.
item_pos	The `Position` of the item in the list.
cur_items	A list of current items at the time of the event that triggered the callback. When the *mode* is OL_NONEXCLUSIVE, this gives a list of all the current items. When the *mode* is OL_EXCLUSIVE or OL_EXCLUSIVE_NONESET, this variable has the same value as the field *item*. This points to temporary storage that the application must copy into its own space if needed.
num_cur_items	The number of current items in the list.
cur_items_pos	An array of integers, representing one for each current item. This points to temporary storage that the application must copy into its own space if needed. This field is currently not implemented in OLIT.

OlSlistItemAttrs

The item data structure `OlSlistItemAttrs` is used in the `OlSlistAddItem()` convenience function to indicate the attributes of interest for a ScrollingList item. See also "OlSlistGetItemAttrs" on page 580 and "OlSlistSetItemAttrs" on page 584.

```
typedef struct {
    long            flags;
    OlDefine        label_type;
    OlStr           item_label;
    XImage          *item_image;
    Boolean         item_sensitive;
    Boolean         item_current;
    unsigned char   item_mnemonic;
    XtPointer       user_data;
} OlSlistItemAttrs, *OlSlistItemAttrsPtr;
```

flags Marks which fields in this structure are defined, as the bitwise inclusive OR of the following:

```
#define OlItemLabelType     (1L << 0)
#define OlItemLabel         (1L << 1)
#define OlItemImage         (1L << 2)
#define OlItemSensitive     (1L << 3)
#define OlItemCurrent       (1L << 4)
#define OlItemMnemonic      (1L << 5)
#define OlItemUserData      (1L << 6)
```

label_type The type of label to display for the Item in the view. It can have one of the values:

OL_STRING for a text label
OL_IMAGE for an image label
OL_BOTH for a text and image label.

item_label The string to display for the Item in the view.

item_image The glyph to be displayed for the item in the view. The type of the value of this member depends on the value of the *label_type* member—if OL_BOTH or OL_IMAGE, a glyph is displayed.

item_sensitive TRUE if the item is sensitive. See also "OlSlistGetItemSensitivity" on page 581.

item_current TRUE if the item is current. See also "OlSlistIsItemCurrent" on page 582.

item_mnemonic A single character that is used as a mnemonic accelerator for keyboard traversal.

user_data A pointer to an area available for application-specific data.

OlListItem Structure

Several of the resources defined for the default API use the following `OlListItem` structure:

```
typedef struct _OlListItem {
    OlDefine        label_type;
    XtPointer       label;
    XImage          *glyph;
    OlBitMask       attr;
    XtPointer       user_data;
    unsigned char   mnemonic;
} OlListItem;
```

label_type — Identifies the type of label to display for the Item in the view. It can have one of the values:
- OL_STRING if the label consists only of text
- OL_IMAGE if the label consists only of an image
- OL_BOTH if the label comprises both text and image

label — Specifies the string to display for the Item in the view.

glyph — Specifies the glyph to be displayed for the item in the view. The type of the value of this member depends on the value of the *label_type* member—if OL_BOTH or OL_IMAGE, a glyph is displayed.

attr — Defines attributes of the Item. It is a 32-bit vector with the field:
- OL_LIST_ATTR_APPL—available for application use. This field consists of the low-order 16 bits, to be used as the application sees fit.
- OL_LIST_ATTR_CURRENT—if the Item is a current item.

Other bit values are defined, but should not be used by the application.

mnemonic — A single character that is used as a mnemonic accelerator for keyboard traversal.

user_data — A pointer to an area available for application-specific data.

OlListToken Structure

In the default API, the ScrollingList widget identifies each item with a "token" of type `OlListToken`. The ScrollingList widget assigns the token when an item is added by the application, and the application uses the token in later references to the item. A zero value is allowed in some contexts where an `OlListToken` is expected, as a way to refer to no item.

As a convenience to the application, the macro `OlListItemPointer(`*`token`*`)` converts an `OlListToken` value into a pointer to the corresponding `OlListItem`. The application can change the values of the `OlListItem` members, but should let the ScrollingList widget know that they have changed, using the `XtNapplTouchItem` routine. No checking is done for incorrect `OlListToken` arguments to the `OlListItemPointer` macro.

The `OlListToken` value can be coerced into the type `caddr_t` and back without loss of precision.

Activation Types

The following table lists the activation types used by the ScrollingList.

Table 9-27 ScrollingList Activation Types

Activation Type	Semantics	Resource Name
OL_ADJUST	ADJUST	XtNadjustBtn
OL_ADJUSTKEY	ADJUST	XtNadjustKey
OL_CANCEL	CANCEL	XtNcancelKey
OL_COPY	COPY	XtNcopyKey
OL_CUT	CUT	XtNcutKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MENU	MENU	XtNmenuBtn
OL_MENUKEY	MENU	XtNmenuKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_MULTIDOWN	JUMP DOWN	XtNmultiDownKey
OL_MULTIUP	JUMP UP	XtNmultiUpKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PAGEDOWN	ALT+PANE DOWN	XtNpageDownKey
OL_PAGEUP	ALT+PANE UP	XtNpageUpKey
OL_PANEEND	PANE DOWN	XtNpaneEndKey
OL_PANESTART	PANE UP	XtNpaneStartKey
OL_PASTE	PASTE	XtNpasteKey

Table 9-27 ScrollingList Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_SCROLLBOTTOM	DATA END	XtNscrollBottomKey
OL_SCROLLDOWN	ALT+DOWN	XtNscrollDownKey
OL_SCROLLTOP	DATA START	XtNscrollTopKey
OL_SCROLLUP	ALT+UP	XtNscrollUpKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

Activation types not described in the following table are described in "Common Activation Types" on page 68.

OL_ADJUST/ OL_ADJUSTKEY

These activation types can be used to make additional list items current. They apply only if the ScrollingList is selectable. If the `XtNtextField` widget is being edited, editing will be terminated and its `XtNverification` callback (if any) will be called. If other list items are already selected (highlighted), they will be cleared. The current item will become the current selection, and will be highlighted if it is not already. If the item has an `XtNuserMakeCurrent` callback, it will be called.

OL_COPY

This activation type will copy the currently selected items to the clipboard as a <newline>-separated list of text items in the order they appear. The OL_LIST_ATTR_CURRENT *attr* field of the currently selected items will be cleared.

OL_CUT

This activation type applies only to ScrollingLists that are editable. The currently selected items are moved to the clipboard as a <newline>-separated list of text items in the order they appear in the ScrollingList. The `XtNuserDeleteItems` callback will be called with an `OlListDelete` structure containing the currently selected items.

OL_MENU/
OL_MENUKEY

These activation types will post the ScrollingList menu.

OL_MULTIDOWN/
OL_PAGEDOWN

These activation types will scroll the list down a number of items equivalent to the current `XtNviewHeight`.

OL_MULTIUP/
OL_PAGEUP

These activation types will scroll the list up a number of items equivalent to the current `XtNviewHeight`.

OL_PANEEND

This activation will type make the last list item the current item.

OL_PANESTART

This activation type will make the first list item the current item.

OL_PASTE

This activation type applies only to an editable ScrollingList. When the `XtNtextField` widget has focus within an editable ScrollingList, then the OL_PASTE activation type will insert the contents of the clipboard at the current insert position.

OL_SCROLLBOTTOM

This activation type will scroll the list so the last item is at the bottom of the view and update the scrollbar appropriately.

OL_SCROLLDOWN

If the list is not already at the bottom, this activation type will scroll the list down one item, updating the list view and scrollbar.

OL_SCROLLTOP

This activation type will scroll the list so the first item is at the top of the view, and updates the scrollbar appropriately.

OL_SCROLLUP

If the list is not already at the top, this activation type will scroll the list up one item, updating the list view and scrollbar.

OL_SELECT/ OL_SELECTKEY

These activation types can be used to make one or more list items current. They apply only if the ScrollingList is selectable. If the `XtNtextField` widget is being edited, editing will be terminated and its `XtNverification` callback (if any) will be called. If other list items are already selected (highlighted), they will be cleared. The item will become the current selection, and will be highlighted if it is not already. If the item has an `XtNuserMakeCurrent` callback, it will be called.

See Also

"Scrollbar Widget" on page 508,
"ScrollingList Functions" on page 578,
"TextField Widget" on page 665.

ScrollingList Functions

The following convenience functions are provided to manipulate the ScrollingList widget in the *new API* mode; see "ScrollingList Modes" on page 552.

OlSlistAddItem

```
#include <Xol/ScrollingL.h>

OlSlistItemPtr OlSlistAddItem(
    Widget                  widget,
    OlSlistItemAttrsPtr     item_attr,
    OlSlistItemPtr          ref_item);
```

`OlSlistAddItem()` adds an item before *ref_item*, given the attributes of the item, and returns a pointer to the item. The value of *ref_item* can be zero to append the new item to the bottom of list. The content of *item_attr* will be copied by the ScrollingList widget into space that it maintains; however, the data pointed to by the *item_image* member will not be copied. The application must use the `OlSlistGetItemAttrs()` and `OlSlistSetItemAttrs()` functions to get and set the data associated with an item and then use the `OlSlistTouchItem()` function to inform the ScrollingList widget that the data has been changed. If mapped and if allowed by the application, the ScrollingList widget updates the view if the new item will be in the view. The view is changed as little as possible: if the new item is in the upper half of the view, the items above it are scrolled up and the top item is scrolled off; if the new item is in the lower half of the view, the items below it are scrolled down and the bottom item is scrolled off.

OlSlistDeleteAllItems

```
#include <Xol/ScrollingL.h>

void OlSlistDeleteAllItems(Widget widget);
```

`OlSlistDeleteAllItems()` deletes all items from the Scrollinglist.

OlSlistDeleteItem

```
#include <Xol/ScrollingL.h>

void OlSlistDeleteItem(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistDeleteItem()` deletes *item* from the ScrollingList.

OlSlistDeleteItems

```
#include <Xol/ScrollingL.h>

void OlSlistDeleteItems(
    Widget              widget,
    OlSlistItemPtr      *items,
    int                 num_items);
```

`OlSlistDeleteItems()` deletes *item*s from the ScrollingList.

OlSlistEditItem

```
#include <Xol/ScrollingL.h>

void OlSlistEditItem(
    Widget              widget,
    Boolean             insert,
    OlSlistItemPtr      ref_item);
```

`OlSlistEditItem()` allows programmatic editing of a ScrollingList item. If *insert* is TRUE, then *ref_item* identifies the item before which a new item is to inserted. If *insert* is TRUE, then *ref_item* can be NULL, in which case a new item would be added to the end of the list. If *insert* is FALSE, then *ref_item* refers to the item that should be edited and must be a valid item in the list.

OlSlistEndEdit

```
#include <Xol/ScrollingL.h>

void OlSlistEndEdit(Widget widget);
```

`OlSlistEndEdit()` should be called by the application when the user has finished editing an item.

OlSlistFirstViewableItem

```
#include <Xol/ScrollingL.h>

void OlSlistFirstViewableItem(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistFirstViewableItem()` sets *item* as the first viewable item in the list.

OlSlistGetItemAttrs

```
#include <Xol/ScrollingL.h>

void OlSlistGetItemAttrs(
    Widget              widget,
    OlSlistItemPtr      item,
    OlSlistItemAttrs    *get_attrs);
```

`OlSlistGetItemAttrs()` returns the item attributes of *item* in *get_attrs*. The *flags* field must be set appropriately in *get_attrs*. The application must not free any members of the *get_attrs* field and must make copies of the data as needed.

OlSlistGetItemImage

```
#include <Xol/ScrollingL.h>

XImage *OlSlistGetItemImage(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistGetItemImage()` returns the image corresponding to *item*, or NULL if the item does not have an associated image.

OlSlistGetItemLabel

```
#include <Xol/ScrollingL.h>

OlStr OlSlistGetItemLabel(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistGetItemLabel()` returns the label corresponding to *item*, or NULL if the item does not have an associated label.

OlSlistGetItemSensitivity

```
#include <Xol/ScrollingL.h>

Boolean OlSlistGetItemSensitivity(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistGetItemSensitivity()` returns TRUE if the *item* is sensitive; otherwise, it returns FALSE.

OlSlistGetItemType

```
#include <Xol/ScrollingL.h>

OlDefine OlSlistGetItemType(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistGetItemType()` returns the item type, given the *item*. Valid values are:

OL_STRING if *item* just has a string
OL_IMAGE if *item* just has a image
OL_BOTH if *item* has both string and image
OL_NONE if *item* is not a valid item in the list

OlSlistGetItemUserData

```
#include <Xol/ScrollingL.h>

XtPointer OlSlistGetItemUserData(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistGetItemUserData()` returns the *item*'s *user_data*, if any.

OlSlistGetMode

```
#include <Xol/ScrollingL.h>

OlDefine OlSlistGetMode(Widget widget);
```

`OlSlistGetMode()` returns the mode of the ScrollingList:

OL_EXCLUSIVE	Exclusive mode; one item is current at all times.

OL_EXCLUSIVE_NONESET	Exclusive-none-set mode; zero or one item is current at all times.
OL_NONEXCLUSIVE	Nonexclusive mode; zero, one, or many items are current.
OL_NONE	Default API mode, for backwards compatibility.

OlSlistGetNextItem

```
#include <Xol/ScrollingL.h>

OlSlistItemPtr OlSlistGetNextItem(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistGetNextItem()` returns the next item, given a valid item. If *item* is the last item in the ScrollingList, then it returns NULL.

OlSlistGetPrevItem

```
#include <Xol/ScrollingL.h>

OlSlistItemPtr OlSlistGetPrevItem(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistGetPrevItem()` returns the previous item, given a valid *item*. If *item* is the first item in the ScrollingList, then it returns NULL.

OlSlistIsItemCurrent

```
#include <Xol/ScrollingL.h>

Boolean OlSlistIsItemCurrent(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistIsItemCurrent()` returns TRUE if the *item* is current; otherwise, it returns FALSE.

OlSlistIsValidItem

```
#include <Xol/ScrollingL.h>

Boolean OlSlistIsValidItem(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistIsValidItem()` returns TRUE if the specified *item* is a valid item in the list; otherwise, it returns FALSE.

OlSlistLastViewableItem

```
#include <Xol/ScrollingL.h>

void OlSlistLastViewableItem(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistLastViewableItem()` sets *item* to be the last viewable item in the list.

OlSlistMakeAllItemsNotCurrent

```
#include <Xol/ScrollingL.h>

void OlSlistMakeAllItemsNotCurrent(Widget widget);
```

`OlSlistMakeAllItemsNotCurrent()` makes all items not current.

OlSlistMakeItemCurrent

```
#include <Xol/ScrollingL.h>

void OlSlistMakeItemCurrent(
    Widget              widget,
    OlSlistItemPtr      item,
    Boolean             issue_callback);
```

`OlSlistMakeItemCurrent()` makes an item current, adds it to the list of current items, and issues a current callback if the `Boolean` value *issue_callback* is TRUE.

OlSlistMakeItemNotCurrent

```
#include <Xol/ScrollingL.h>

void OlSlistMakeItemNotCurrent(
    Widget              widget,
    OlSlistItemPtr      item,
    Boolean             issue_callback);
```

`OlSlistMakeItemNotCurrent()` makes an *item* not current in the ScrollingList and also removes it from the current list. It issues an `XtNitemNotCurrentCallback` if *issue_callback* is TRUE.

OlSlistSetItemAttrs

```
#include <Xol/ScrollingL.h>

void OlSlistSetItemAttrs(
    Widget              widget,
    OlSlistItemPtr      item,
    OlSlistItemAttrs    *set_attrs);
```

`OlSlistSetItemAttrs()` sets the attributes pointed to in *set_attrs* on the specified *item*. The *flags* field must be appropriately set in *set_attrs*. The widget then copies this data into space that it maintains. The data pointed to by the *item_image* member of *set_attrs* will not be copied.

OlSlistTouchItem

```
#include <Xol/ScrollingL.h>

void OlSlistTouchItem(
    Widget              widget,
    OlSlistItemPtr      item);
```

`OlSlistTouchItem()` should be used by the application when it changes the attributes of an item. If mapped and if allowed by the application (see "OlSlistUpdateView" on page 585), the ScrollingList widget updates the view if the changed *item* is visible.

OlSlistUpdateView

```
#include <Xol/ScrollingL.h>

void OlSlistUpdateView(
    Widget      widget,
    Boolean     ok);
```

`OlSlistUpdateView()` can be called by the application to keep the ScrollingList widget from updating the view or to let it again update the view. If *ok* is TRUE, the ScrollingList widget can update the View as it changes; otherwise, it cannot update the view.

OlSlistViewItem

```
#include <Xol/ScrollingL.h>

void OlSlistViewItem(
    Widget widget,
     OlSlistItemPtr item);
```

`OlSlistViewItem()` can be called by the application to place a particular ScrollingList item in view. See description of the `XtNapplViewItem` resource in the ScrollingList widget section for more details.

Known Deficiencies

There is no programmatic interface equivalent to the `XtNapplViewItem` resource (such as `OlSlistUpdateView()`'s relationship to `XtNapplUpdateView`). The following workaround is available:

```
OlApplViewItemProc ViewItem;
OlSlistItemPtr item;          /* the item to be in view */

...

XtVaGetValues(slist, XtNapplViewItem, &ViewItem, NULL);
(*ViewItem)(slist, item);     /* this brings 'item' into view */
```

9

Slider Widget

Slider Widget

Class

Class Name: `Slider`
Class Pointer: `sliderWidgetClass`

Ancestry

Core-Primitive-Slider

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/Slider.h>
```

Description

The Slider widget enables the user to set a numeric value graphically.

Components

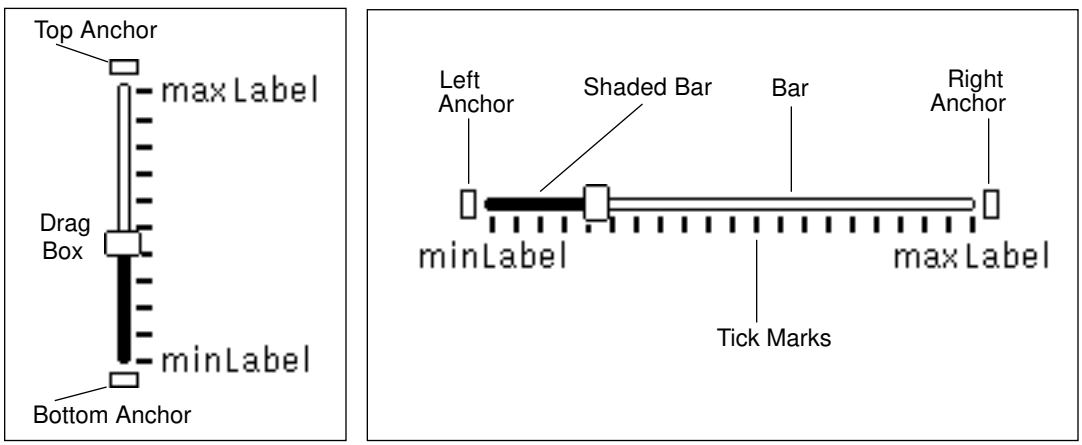

Figure 9-14 Vertical and Horizontal Sliders

A Slider consists of the following elements:

- Top (left) anchor
- Bottom (right) anchor
- Drag box
- Bar (typically)
- Shaded bar (typically)
- Current value (not visible)
- Minimum value (not visible)
- Maximum value (not visible)

The current value is the numeric value a user attempts to change with the Slider widget.

Drag Box Motion

As visual feedback to the user, the drag box moves up or down (or left or right) along the bar as the current value changes.

Dragging SELECT

The user can change the current value by dragging the drag box with SELECT. The pressing of SELECT must start with the pointer in the drag box, but the drag box (and the current value) track the pointer motion regardless of where it goes while SELECT is pressed. This means it is not possible for the user to change the current value by first pressing SELECT outside the drag box and then moving the pointer into it. Only the component of the pointer motion parallel to the Bar is tracked, and the motion of the drag box (and change in the current value) are limited by the length of the Bar.

Clicking SELECT

Clicking SELECT above the drag box for a vertical slider, or to the right for a horizontal slider, increases the current value by an application-specified amount, moves the drag box to correspond to the new current value, and moves the pointer to keep it on the drag box.

Clicking SELECT to the other side of the drag box decreases the value by the same amount and moves the drag box and pointer accordingly.

Pressing SELECT repeats this action.

Moving Drag Box to Limits

Clicking SELECT on one of the bottom/left or top/right anchors causes the current value to take on the minimum value or maximum value, respectively, and moves the drag box to the limit in the direction of the Anchor. If the drag box is already at the limit, nothing happens.

Clicking SELECT on an anchor highlights the anchor while the current value is changed.

Application Notification

The application gets notified about a change in the current value on the release of SELECT for either the drag or click. It is responsible for providing any feedback to the user deemed appropriate, such as updating the current value in a text field.

Coloration

For 3D and 2D, the area surrounding the Slider and its labels is drawn with the parent's `XtNbackground`. `XtNfontColor` is used to draw the minimum and maximum labels.

For 3D, the slider, endbox, and tickmark coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1. `XtNforeground` is used to draw the current-value indicator bar on the Slider.

For 2D, `XtNbackground` and `XtNforeground` are used to render the slider, endboxes, and tickmarks as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

If the toolkit resource `XtNmouseless` is set to TRUE and the toolkit resource `XtNinputFocusFeedback` is set to OL_INPUT_FOCUS_COLOR, then the background of the Slider will be drawn with the value of `XtNinputFocusColor` when it receives input focus. However, if `XtNinputFocusColor` is the same as `XtNbackground`, then the Slider inverts `XtNforeground` and `XtNbackground`. Once the input focus leaves the Slider, the original coloration is restored.

Keyboard Traversal

The Slider's default value of the `XtNtraversalOn` resource is TRUE.

The user can operate the Slider by using the keyboard to move the Drag Box and access the Anchors. The following keys manipulate the Current Value:

- SCROLLUP and SCROLLRIGHT increase the Current Value by an application-specified amount, and move the Drag Box to correspond to the new Current Value.
- SCROLLDOWN and SCROLLLEFT decrease the Current Value by an application-specified amount, and move the Drag Box to correspond to the new Current Value.
- SCROLLTOP and SCROLLRIGHTEDGE cause the Current Value to take on the Maximum Value, and move the Drag Box to a vertical slider's top anchor or a horizontal slider's right anchor. The anchor is briefly highlighted while the Current Value is changed and the Drag Box is moved.
- SCROLLBOTTOM and SCROLLLEFTEDGE cause the Current Value to take on the Minimum Value, and move the Drag Box to a vertical slider's bottom anchor or a horizontal slider's left anchor. The anchor is briefly highlighted while the Current Value is changed and the Drag Box is moved.

The Slider widget responds to the following keyboard navigation keys:

- NEXTFIELD, MOVEDOWN, and MOVERIGHT move to the next traversable widget in the window
- PREVFIELD, MOVEUP, and MOVELEFT move to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application

Keyboard Mnemonic Display

The Slider does not display the mnemonic accelerator. If the Slider is the child of a Caption widget, the Caption widget will display the mnemonic as part of the label.

Keyboard Accelerator Display

The Slider does not respond to a keyboard accelerator because the results of clicking the SELECT button on a Slider depends on the pointer position. So, the Slider does not display a keyboard accelerator.

Resources

Table 9-28 Slider Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-29 Slider Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID

Table 9-29 Slider Primitive Resources (Continued)

Name	Type	Default	Access
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	SGI
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 9-30 Slider Resources

Name	Type	Default	Access
XtNdragCBType	OlDefine	OL_CONTINUOUS	SGI
XtNendBoxes	Boolean	TRUE	SGI
XtNgranularity	int	1	SGI
XtNinitialDelay	int	500	SGI
XtNmaxLabel	OlStr	NULL	SGI
XtNminLabel	OlStr	NULL	SGI
XtNorientation	OlDefine	OL_VERTICAL	GI
XtNpointerWarping	Boolean	TRUE	SGI
XtNrecomputeSize	Boolean	FALSE	SGI
XtNrepeatRate	int	100	SGI
XtNsliderMax	int	100	SGI
XtNsliderMin	int	0	SGI
XtNsliderMoved	XtCallbackList	NULL	SGIO
XtNsliderValue	int	0	SGI
XtNspan	Dimension	OL_IGNORE	SGI
XtNstopPosition	OlDefine	OL_ALL	SGI
XtNticks	int	0	SGI
XtNtickUnit	OlDefine	OL_NONE	SGI
XtNuseSetValCallback	Boolean	FALSE	SGI

XtNdragCBType

Class	Type	Default	Access
XtCDragCBType	OlDefine	OL_CONTINUOUS	SGI

Synopsis: The frequency of issuing `XtNsliderMoved` callbacks during a drag operation.

Values: `OL_CONTINUOUS/"continuous"` – Issue callbacks continuously.
`OL_GRANULARITY/"granularity"` – Issue callbacks only when the drag box crosses any granularity positions.
`OL_RELEASE/"release"` – Issue callbacks only once when the SELECT button is released.

XtNendBoxes

Class	Type	Default	Access
XtCEndBoxes	Boolean	TRUE	SGI

Synopsis: Whether the end boxes are displayed.

Values: `TRUE/"true"` – Display end boxes.
`FALSE/"false"` – Do not display end boxes.

XtNgranularity

Class	Type	Default	Access
XtCGranularity	int	1	SGI

Synopsis: How much clicking SELECT changes the current value.

Values: 1 ≤ `XtNgranularity` ≤ (`XtNsliderMax – XtNsliderMin`)

Clicking SELECT on the bar or shaded bar attempts to change the current value by the amount given in this resource. Dragging the drag box with SELECT changes the current value by this amount before the `XtNsliderMoved` callbacks are issued.

XtNinitialDelay

Class	Type	Default	Access
XtCInitialDelay	int	500	SGI

Synopsis: The time, in milliseconds, before the first action occurs when SELECT is pressed on the bar or shaded bar.

Values: 0 < `XtNinitialDelay`

XtNmaxLabel

Class	Type	Default	Access
XtCLabel	OlStr	NULL	SGI

Synopsis: The label to be placed next to the maximum value position.
Values: Any `OlStr` value valid in the current locale.

For a vertical slider, the label will be placed to the right of the minimum value position. If there is not enough space for the entire label and `XtNrecomputeSize` is FALSE, the label will be truncated from the right end. If there is not enough space for the entire label and `XtNrecomputeSize` is TRUE, then the widget will request more space to show the entire label.

For a horizontal slider, the label will be placed centered and below the maximum value position. If there is not enough room to center the label and `XtNrecomputeSize` is set to FALSE, the right end of the label will be aligned with the right anchor. If this label collides with the `XtNminLabel`, some part of the labels will overlap. If there is not enough room to center the label and `XtNrecomputeSize` is set to TRUE, the widget will request more space to center the label below the maximum value position.

XtNminLabel

Class	Type	Default	Access
XtCLabel	OlStr	NULL	SGI

Synopsis: The label to be placed next to the minimum value position.
Values: Any `OlStr` value valid in the current locale.

For a vertical slider, the label is placed to the right of the minimum value position. If there is not enough space for the entire label and `XtNrecomputeSize` is FALSE, the label will be truncated from the right end. If there is not enough space for the entire label and `XtNrecomputeSize` is TRUE, then the widget will request more space to show the entire label.

For a horizontal slider, the label will be placed centered and below the minimum value position. If there is not enough room to center the label and `XtNrecomputeSize` is set to FALSE, the beginning of the label will be aligned with the left anchor and will be drawn to the right. If this label collides with the `XtNmaxLabel`, some part of the labels will overlap. If there is not enough room to center the label and `XtNrecomputeSize` is set to TRUE, the widget will request more space to center the label below the minimum value position.

XtNorientation

Class	Type	Default	Access
XtCOrientation	OlDefine	OL_VERTICAL	GI

Synopsis: The direction for the visual presentation of the widget.
Values: `OL_HORIZONTAL/"horizontal"` – Define a horizontal slider.
`OL_VERTICAL/"vertical"` – Define a vertical slider.

XtNpointerWarping

Class	Type	Default	Access
XtCPointerWarping	Boolean	TRUE	SGI

Synopsis: Whether the pointer will warp; i.e., follow the drag box as it moves.
Values: `TRUE/"true"` – The pointer will warp.
`FALSE/"false"` – The pointer will not warp.

XtNrecomputeSize

Class	Type	Default	Access
XtCRecomputeSize	Boolean	FALSE	SGI

Synopsis: Whether the Slider widget should resize itself whenever needed, to compensate for the space needed to show the tick marks and the labels.
Values: `TRUE/"true"` – The Slider will resize itself.
`FALSE/"false"` – The Slider will not resize itself.

The Slider widget uses the value of `XtNspan`, the sizes of the labels, and `XtNtickUnit` to determine the preferred size.

XtNrepeatRate

Class	Type	Default	Access
XtCRepeatRate	int	100	SGI

Synopsis: The time, in milliseconds, between repeated actions when SELECT is pressed on the bar or shaded bar.
Values: 0 < `XtNrepeatRate`

XtNsliderMax/ XtNsliderMin

Class	Type	Default	Access
XtCSliderMax	int	100	SGI
XtCSliderMin	int	0	SGI

Synopsis: The range of values tracked by the Slider widget.
Values: `XtNsliderMin` < `XtNsliderMax`

Mathematically, the range is closed on both ends; that is, the range is the following subset of the set of integers:

$$\texttt{XtNsliderMin} \leq \textit{range} \leq \texttt{XtNsliderMax}$$

This is independent of the drag box displayed in the Slider widget. The Slider widget takes into account the size of the drag box when relating the physical range of movement to the range of values.

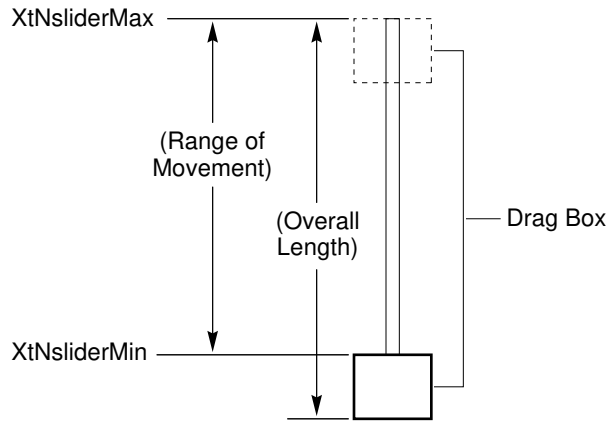

Figure 9-15 Drag Box Range of Movement

XtNsliderMoved

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the Slider widget is manipulated.

The *call_data* parameter is a pointer to a structure of the form:

```
typedef struct OlSliderVerify {
    int         new_location;
    Boolean     more_cb_pending;
} OlSliderVerify;
```

new_location The new location of the Slider.

more_cb_pending Specifies if there are more callbacks pending.

An `XtGetValues()` inside the callback will return the previous value.

XtNsliderValue

Class	Type	Default	Access
XtCSliderValue	int	0	SGI

Synopsis: The current position of the drag box.
Values: `XtNsliderMin` ≤ `XtNsliderValue` ≤ `XtNsliderMax`

The Slider widget keeps this resource up to date; however, an application can also get the current value through the `XtNsliderMoved` callbacks.

XtNspan

Class	Type	Default	Access
XtCSpan	Dimension	OL_IGNORE	SGI

Synopsis: If `XtNrecomputeSize` is set to TRUE, then `XtNspan` should be set to reflect the preferred length of the slider, not counting the space needed for the labels. The Slider widget uses the span value, the sizes of the labels, and `XtNtickUnit` to determine the preferred size.

XtNstopPosition

Class	Type	Default	Access
XtCStopPosition	OlDefine	OL_ALL	SGI

Synopsis: The behavior of the drag box at the end of a drag operation.
Values: `OL_ALL/"all"` – At the end of a drag operation, the drag box will be positioned where it stops.
`OL_GRANULARITY/"granularity"` – The drag box will snap to the nearest granularity position.
`OL_TICKMARK/"tickmark"` – The drag box will snap to the nearest tickmark position.

XtNticks

Class	Type	Default	Access
XtCTicks	int	0	SGI

Synopsis: The interval between tick marks.
Values: The unit of the interval value is determined by `XtNtickUnit`.

XtNtickUnit

Class	Type	Default	Access
XtCTickUnit	OlDefine	OL_NONE	SGI

Synopsis: The number of units of the interval between tick marks.
Values: `OL_NONE/"none"` – The widget will display no tick marks and will ignore `XtNticks`.
`OL_SLIDERVALUE/"slidervalue"` – The widget will interpret `XtNticks` as the same unit as the slider value.
`OL_PERCENT/"percent"` – The widget will interpret `XtNticks` as the percentage of the slider value range.

XtNuseSetValCallback

Class	Type	Default	Access
XtCUseSetValCallback	Boolean	FALSE	SGI

Synopsis: Whether a `XtNsliderMoved` callback is called when Slider is manipulated by doing an `XtSetValues()` call.
Values: `TRUE/"true"` – The callback will be called.
`FALSE/"false"` – The callback will not be called.

Activation Types

The following table lists the activation types used by the Slider.

Table 9-31 Slider Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey

Table 9-31 Slider Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_SCROLLBOTTOM	DATA END	XtNscrollBottomKey
OL_SCROLLDOWN	ALT+DOWN	XtNscrollDownKey
OL_SCROLLLEFT	ALT+LEFT	XtNscrollLeftKey
OL_SCROLLLEFTEDGE	ALT+{	XtNscrollLeftEdge
OL_SCROLLRIGHT	ALT+RIGHT	XtNscrollRightKey
OL_SCROLLRIGHTEDGE	ALT+}	XtNscrollRightEdge
OL_SCROLLTOP	DATA START	XtNscrollTopKey
OL_SCROLLUP	ALT+UP	XtNscrollUpKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

Activation types not described in the following table are described in "Common Activation Types" on page 68.

OL_SCROLLBOTTOM

For a scrollbar with `XtNorientation` of OL_VERTICAL, this activation type will move the slider to the value of `XtNsliderMin` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLDOWN/ OL_SCROLLLEFT

These activation types will move the slider one negative unit of granularity and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLLEFTEDGE

For a slider with `XtNorientation` of OL_HORIZONTAL, this activation type will move the slider to the value of `XtNsliderMin` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLRIGHTEDGE

For a slider with `XtNorientation` of OL_HORIZONTAL, this activation type will move the slider to the value of `XtNsliderMax` minus the value of `XtNproportionLength` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLTOP

For a slider with `XtNorientation` of OL_VERTICAL, this activation type will move the slider to the value of `XtNsliderMax` and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SCROLLUP/ OL_SCROLLRIGHT

These activation types will move the slider one positive unit of granularity and call the `XtNsliderMoved` callback with the appropriate `OlScrollbarVerify` structure.

OL_SELECT

This activation type depends on the position of the pointer within the Slider widget. When the pointer is positioned on the drag box, the `XtNsliderMoved` callback will be called according to the value of the `XtNdragCBType`. When the pointer is positioned on the right anchor, the behavior will be the same as the OL_SCROLLRIGHTEDGE activation type. When the pointer is positioned on the top anchor, the behavior will be the same as the OL_SCROLLTOP activation type. When the pointer is positioned on the left anchor, the behavior will be the same as the OL_SCROLLLEFTEDGE activation type. When the pointer is positioned on the bottom anchor, the behavior will be the same as the OL_SCROLLBOTTOM activation type. When the pointer is positioned on the bar above or to the right of the drag box, the behavior will be the same as the OL_SCROLLUP and OL_SCROLLRIGHT activation type. When the pointer is positioned on the below or to the left of the drag box, the behavior will be the same as the OL_SCROLLLEFT and OL_SCROLLDOWN activation type.

See Also

"Gauge Widget" on page 395,
"Scrollbar Widget" on page 508.

9

StaticText Widget

Class

Class Name: `StaticText`
Class Pointer: `staticTextWidgetClass`

Ancestry

Core-Primitive-StaticText

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/StaticText.h>
```

Description

The StaticText widget provides a way to present an uneditable block of single or multi-line text to the user. The layout of the text is configured by a few simple layout controls.

Word Wrap

If the text is too long to fit in the width provided by the StaticText widget, the text may be "wrapped" if the application requests it. The wrapping occurs at a space between words, if possible, leaving as many words on a line as will fit. If a word is too long for the width, it is wrapped between characters. An embedded newline will always cause a wrap.

Text Clipping

If the text is not wrapped, it will be truncated if it cannot fit in the width of the StaticText widget. The application can choose whether the truncation occurs on the left, right, or evenly on both sides of each line of the text.

If the text is too large to fit in the height provided by the StaticText widget, the text is clipped on the bottom. The clipping falls on a pixel boundary, not between lines, so that it is possible that only the upper part of the last line of text may be visible.

Space Stripping

The application can choose to have leading spaces or trailing spaces (or both) stripped from the text before display, or can choose to have no stripping done.

Selecting and Operating on Text

The StaticText widget allows text to be selected in several ways and then copied. See "Text Selection Operations" on page 204 for the description of these operations. The application can control whether or not this selectability is allowed on a StaticText widget.

Coloration

The diagram illustrates the resources that affect StaticText coloration.

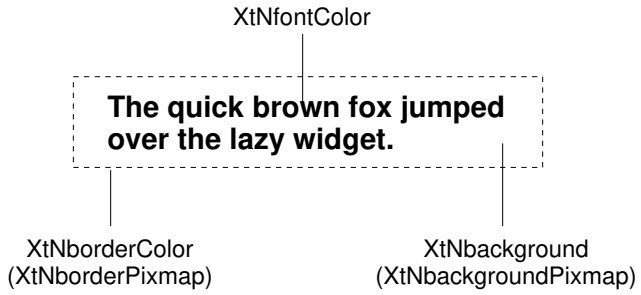

Figure 9-16 StaticText Coloration

Keyboard Traversal

The default value of the `XtNtraversalOn` resource is FALSE.

If `XtNtraversalOn` is set to TRUE, the widget responds to the following keyboard navigation keys:

- NEXTFIELD moves to the next traversable widget in the window
- PREVFIELD moves to the previous traversable widget in the window
- NEXTWINDOW moves to the next window in the application
- PREVWINDOW moves to the previous window in the application
- NEXTAPP moves to the first window in the next application
- PREVAPP moves to the first window in the previous application

Keyboard Mnemonic and Accelerator Display

The StaticText does not have keyboard mnemonic or keyboard accelerator capabilities.

Resources

Table 9-32 StaticText Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	(calculated)	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	(calculated)	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-33 StaticText Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	n/a
XtNacceleratorText	String	NULL	n/a
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID

Table 9-33 StaticText Primitive Resources (Continued)

Name	Type	Default	Access
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	n/a
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 9-34 StaticText Resources

Name	Type	Default	Access
XtNalignment	OlDefine	OL_LEFT	SGI
XtNgravity	int	CenterGravity	SGI
XtNhSpace	Dimension	2	SGI
XtNlineSpace	int	0	SGI
XtNrecomputeSize	Boolean	TRUE	SGI
XtNselectable	Boolean	TRUE	SGI
XtNstring	OlStr	NULL	SGI
XtNstrip	Boolean	TRUE	SGI
XtNvSpace	Dimension	2	SGI
XtNwrap	Boolean	TRUE	SGI

XtNalignment

Class	Type	Default	Access
XtCAlignment	OlDefine	OL_LEFT	SGI

Synopsis: The alignment of the lines to be applied when drawing the text.
Values: `OL_LEFT/"left"` – The left sides of lines are vertically aligned.
`OL_CENTER/"center"` – The centers are vertically aligned.
`OL_RIGHT/"right"` – The right sides are vertically aligned.

XtNgravity

Class	Type	Default	Access
XtCGravity	Int	CenterGravity	SGI

Synopsis: How extra space is used with the StaticText widget. See Table 9-35.

If `XtNrecomputeSize` is set to FALSE, the application can set a width and height to the StaticText widget that exceeds the size needed to display the string. This resource controls the use of any extra space.

Table 9-35 XtNgravity Values

Value	Action
CenterGravity	String is centered vertically and horizontally in the extra space.
NorthGravity	Top edge of the string is aligned with the top edge of the space and centered horizontally.
SouthGravity	Bottom edge of the string is aligned with the bottom edge of the space and centered horizontally.
EastGravity	Right edge of the string is aligned with the right edge of the space and centered vertically.
WestGravity	Left edge of the string is aligned with the left edge of the space and centered vertically.
NorthWestGravity	Top and left edges of the string are aligned with the top and left edges of the space.
NorthEastGravity	Top and right edges of the string are aligned with the top and right edges of the space.
SouthWestGravity	Bottom and left edges of the string are aligned with the bottom and left edges of the space.
SouthEastGravity	Bottom and right edges of the string are aligned with the bottom and right edges of the space.

XtNheight

Class	Type	Default	Access
XtCHeight	Dimension	(calculated)	SGI

Synopsis: The height of the StaticText widget, not including the border.

If `XtNrecomputeSize` is set to TRUE, `XtNheight` is set to:

(*height of text*) + 2 × `XtNhSpace`.

XtNhSpace/ XtNvSpace

Class	Type	Default	Access
XtCHSpace	Dimension	2	SGI
XtCVSpace	Dimension	2	SGI

Synopsis: The number of pixels of horizontal (vertical) padding between the window borders and the displayed string.
Values: 0 ≤ `XtNhSpace`
0 ≤ `XtNhSpace`

XtNlineSpace

Class	Type	Default	Access
XtCLineSpace	int	0	SGI

Synopsis: The amount of space between lines of text.
Values: –100 ≤ `XtNlineSpace`

The spacing is specified as a percentage of the font height, and is the distance between the baseline of one text line and the top of the next font line. Thus, the distance between successive text baselines, in percentage of the font height, is `XtNlineSpace` + 100.

XtNrecomputeSize

Class	Type	Default	Access
XtCRecomputeSize	Boolean	TRUE	SGI

Synopsis: Whether the StaticText widget should calculate its size and automatically set the `XtNheight` and `XtNwidth` resources.
Values: `TRUE/"true"` – The StaticText widget will do normal size calculations that may cause its geometry to change.
`FALSE/"false"` – The StaticText widget will leave its size alone; this may cause truncation of the visible image being shown by the StaticText widget if the fixed size is too small, or may cause centering if the fixed size is too large.

XtNselectable

Class	Type	Default	Access
XtCSelectable	Boolean	TRUE	SGI

Synopsis: Whether the user can select text in the widget.

Values: TRUE/"true" – The user can select text.
FALSE/"false" – The user cannot select text. SELECT or ADJUST mouse actions and the COPY key action will have no effect.

XtNstring

Class	Type	Default	Access
XtCString	OlStr	NULL	SGI

Synopsis: The (NULL terminated) string to be drawn.
Values: Any OlStr value valid in the current locale.

XtNstrip

Class	Type	Default	Access
XtCStrip	Boolean	TRUE	SGI

Synopsis: How leading and trailing spaces are stripped during the layout of the text string.
Values: TRUE/"true", FALSE/"false", as shown in the following table.

XtNstrip	XtNalignment	Spaces stripped
TRUE	OL_LEFT	Leading spaces stripped.
	OL_RIGHT	Trailing spaces stripped.
	OL_CENTER	Both leading and trailing spaces stripped
FALSE	any	None.

XtNwidth

Class	Type	Default	Access
XtCWidth	Dimension	(calculated)	SGI

Synopsis: The width of the StaticText widget, not including the border.

If XtNrecomputeSize is set to TRUE, XtNwidth is set to:
(*width of text*) + 2 × XtNvSpace.

XtNwrap

Class	Type	Default	Access
XtCWrap	Boolean	TRUE	SGI

Synopsis: How lines that are too long to fit in the width of the StaticText widget are wrapped.

Values: `TRUE/"true", FALSE/"false"`, as shown in the following table. .

XtNwrap	XtNalignment	Wrap action
FALSE	OL_LEFT	Clipped on the right
	OL_RIGHT	Clipped on the left
	OL_CENTER	Clipped equally on both left and right
TRUE	any	Long text is broken between words with each line of the displayed text having as many words as can fit.

Activation Types

The following table lists the activation types used by the StaticText.

Table 9-36 StaticText Activation Types

Activation Type	Semantics	Resource Name
OL_ADJUST	ADJUST	XtNadjustBtn
OL_CANCEL	CANCEL	XtNcancelKey
OL_COPY	COPY	XtNcopyKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_SELECT	SELECT	XtNselectBtn
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

Activation types not described in the following list are described in "Common Activation Types" on page 68.

OL_ADJUST

This activation type can be used to alter a selection, as described in the *OPEN LOOK GUI Functional Specification* section "Adjusting a Text Selection" in Chapter 17.

OL_COPY

This activation type can be used to copy the currently selected text to the clipboard, as described in the *OPEN LOOK GUI Functional Specification* section "Copying Text" in Chapter 17.

OL_SELECT

This activation type can be used to select characters, as described in the *OPEN LOOK GUI Functional Specification* section "Selecting Text" in Chapter 17.

See Also

"TextField Functions" on page 686,
"Text Selection Operations" on page 204,
"TextEdit Functions" on page 660,
"TextEdit Widget" on page 623,
"TextField Widget" on page 665.

Stub Widget

Class

Class Name: `Stub`
Class Pointer: `stubWidgetClass`

Ancestry

Core-Primitive-Stub

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/Stub.h>
```

Description

The Stub widget is a method-driven widget that allows the application to specify procedures at creation and/or `XtSetValues()` time that are normally restricted to a widget's class part.

Most class part procedures have been attached to the instance part. For example, with the Stub widget, it is possible to set the procedure that is called whenever an exposure occurs. It is also possible to set the *setValues* and *initialize* procedures.

Local Widgets

By allowing the application to specify procedures outside the widget class structure, applications can use the Stub widget to build local widgets without having going through the formal steps. For example, suppose an application wanted to create a menu separator widget that inherits its parent's background color at creation time. It would be wasteful to create a new widget to perform these trivial tasks. Instead, the application would use a Stub widget and specify an Initialize procedure for it.

Graphics Applications

The Stub widget can be used to implement graphics applications. Since the application has direct access to the widget's internal expose procedure, the application can take advantage of the exposure compression provided with the *region* argument. This field is not accessible if the application used an Event Handler to trap exposures. Also, since the application has access to the SetValues and SetValuesHook procedures, the application can programmatically modify graphic-related resources of the Stub widget.

Inheriting Procedures from Existing Widgets

Once a Stub widget is created, other Stub widgets can inherit its methods without the application having to specify them again. All the application has to do is specify a reference Stub widget in the creation Arg list and the new Stub widget will inherit all instance methods from the referenced Stub widget.

Wrapping Widgets Around Existing Windows

The Stub widget also allows the application to give widget functionality to existing X windows. For example, if the application wanted to track button presses on the root window, the application would create a Stub widget using the RootWindow ID as the `XtNwindow` resource. Once this has been done, the application can monitor events on the root window by attaching event handlers to the Stub widget.

Coloration

The diagram illustrates the resources used for Stub coloration.

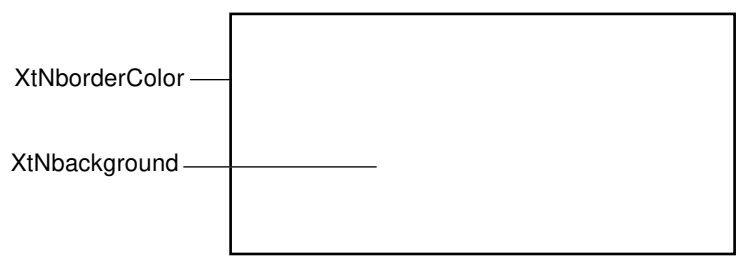

Figure 9-17 Stub Coloration

Keyboard Traversal

The Stub is a Primitive widget and it inherits the translations for traversal actions from the `Primitive` class. The user of a Stub widget should add translations for dealing with the navigation events listed in the section of "TextField Widget" on page 665 that apply to the particular use of the Stub.

Keyboard Mnemonic Display

The Stub does not display the mnemonic accelerator. If the Stub is the child of a Caption widget, the Caption widget can be used to display the Stub's mnemonic.

Keyboard Accelerator Display

The Stub does not display the keyboard accelerator. If the Stub is the child of a Caption widget, the Caption widget can be used to display the Stub's accelerator as part of the label.

Coloration

The Stub widget should be able to display a state that indicates it has input focus. The general heuristic used for this display by OPEN LOOK widgets is that the background color is replaced with the input focus color found in the resource `XtNinputFocusColor`.

Resources

Table 9-37 Stub Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI

Table 9-37 Stub Core Resources (Continued)

Name	Type	Default	Access
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	0	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	0	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 9-38 Stub Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	SGI
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	Red	SGID
XtNmnemonic	unsigned char	'\0'	SGI
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	FALSE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 9-39 Stub Resources

Name	Type	Default	Access
XtNacceptFocusFunc	Function	NULL	SGI
XtNactivateFunc	Function	NULL	SGI
XtNdestroy	Function	NULL	SGI

Table 9-39 Stub Resources (Continued)

Name	Type	Default	Access
XtNexpose	Function	NULL	SGI
XtNgetValuesHook	Function	NULL	SGI
XtNhighlightHandlerProc	Function	NULL	SGI
XtNinitialize	Function	(private)	GI
XtNinitializeHook	Function	NULL	GI
XtNqueryGeometry	Function	NULL	SGI
XtNrealize	Function	(private)	SGI
XtNreferenceStub	Widget	NULL	GI
XtNregisterFocusFunc	Function	NULL	SGI
XtNresize	Function	NULL	SGI
XtNsetValues	Function	NULL	SGI
XtNsetValuesAlmost	Function	(superclass)	SGI
XtNsetValuesHook	Function	NULL	SGI
XtNtraversalHandlerFunc	Function	NULL	SGI
XtNwindow	Window	NULL	GI

XtNacceptFocusFunc

Class	Type	Default	Access
XtCAcceptFocusFunc	Function	NULL	SGI

Synopsis: The function that allows applications to override the default accept focus procedure.

This procedure has the same semantics as the `XtAcceptFocusFunc` Core Widget Class procedure and it is called by the Stub Widget Class's accept focus class procedure. When overriding the default accept focus procedure, the routine `OlCanAcceptFocus()` can be used to check the widget's focus-taking eligibility. `OlSetInputFocus()` (see page 151) should be used instead of `XSetInputFocus()` when explicitly setting focus to a window.

XtNactivateFunc

Class	Type	Default	Access
XtCActivateFunc	Function	NULL	SGI

Synopsis: The procedure called whenever `OlActivateWidget()` is called with the Stub widget ID for which this function was assigned.

The procedure has the following declaration:

```
Boolean activateFunc(
    Widget          w,
    OlVirtualName   activation_type,
    XtPointer       data);
```

If the *activation_type* is valid, the routine should take the appropriate action and return TRUE; otherwise, the routine should return FALSE.

XtNdestroy

Class	Type	Default	Access
XtCDestroy	Function	NULL	SGI

Synopsis: The procedure called when this Stub instance is destroyed.

The procedure has the following declaration:

```
void destroy(Widget w);
```

XtNexpose

Class	Type	Default	Access
XtCExpose	Function	NULL	SGI

Synopsis: The procedure called when the corresponding Stub widget receives an exposure event.

The procedure has the following declaration:

```
void expose(
    Widget      w,
    XEvent      *xevent,
    Region      region);
```

Since the Stub widget class has requested exposure compression, the region field is valid.

XtNgetValuesHook

Class	Type	Default	Access
XtCGetValuesHook	Function	NULL	SGI

Synopsis: The procedure called whenever the application makes an `XtGetValues()` call on the corresponding Stub widget.

The procedure has the following declaration:

```
void getValuesHook(
    Widget      w,
    ArgList     args,
    Cardinal    *num_args);
```

XtNheight

Class	Type	Default	Access
XtCHeight	Dimension	0	SGI

Synopsis: The height of the widget.

If `XtNwindow` has a NULL value, the application must ensure that the dimensions of `XtNwidth` and `XtNheight` are non-NULL. The application can specify the width and height with an Arg list or specify an *initialize* procedure that sets them with non-NULL values. If either of these dimensions is NULL when the application attempts to realize the Stub widget, an error will result.

XtNhighlightHandlerProc

Class	Type	Default	Access
XtCHighlightHandlerProc	Function	NULL	SGI

Synopsis: Whether the Stub widget displays a visual indication that it has, or does not have, input focus.

The `XtNhighlightHandlerProc` procedure will be called to notify a Stub widget that input focus has either moved to or from the widget. The `XtNhighlightHandlerProc` will be called after the widget has lost or accepted focus as a result of calling `OlSetInputFocus()`, calling `OlMoveFocus()`, or matching a KeyPress event to a keyboard traversal command (see "OlMoveFocus" on page 152). The procedure has the following declaration:

```
void highlightHandlerProc(
    Widget      widget,
    OlDefine    highlight_type);
```

This procedure has the responsibility of displaying the object in the appropriate state based on the *highlight_type*. Whenever a Stub widget receives focus, its `XtNhighlightHandlerProc` will be called with a *highlight_type* of OL_IN. An OL_OUT *highlight_type* is used to signal that the Stub widget has lost focus.

XtNinitialize

Class	Type	Default	Access
XtCInitialize	Function	(private)	SGI

Synopsis: The procedure called by XtCreateWidget() for a Stub widget instance.

The procedure has the following declaration:

```
void initialize(
    Widget      request,
    Widget      new,
    ArgList     args,
    Cardinal    *num_args);
```

If the application supplies its own initialize procedure, it is the application's responsibility to deal with the XtNwindow resource. When the XtNwindow resource is non-NULL, the default initialize procedure fills in the Stub widget's XtNx, XtNy, XtNwidth, and XtNheight resource values with the corresponding window attribute values obtained from the window specified by the XtNwindow resource.

XtNinitializeHook

Class	Type	Default	Access
XtCInitializeHook	Function	NULL	SGI

Synopsis: The procedure called by XtCreateWidget() for a Stub widget instance after the *initialize* procedure has been called.

The procedure has the following declaration:

```
void initializeHook(
    Widget      w,
    ArgList     args,
    Cardinal    *num_args);
```

The application can access the creation argument list through this routine. The widget specified with the *w* argument is the *new* widget from the *initialize* procedure.

XtNqueryGeometry

Class	Type	Default	Access
XtCQueryGeometry	Function	NULL	SGI

Synopsis: The procedure called whenever the application does an `XtQueryGeometry()` request on the corresponding Stub widget.

The procedure has the following declaration:

```
void queryGeometry(
    Widget              w,
    XtWidgetGeometry    *request,
    XtWidgetGeometry    *preferred_return);
```

XtNrealize

Class	Type	Default	Access
XtCRealize	Function	(private)	SGI

Synopsis: The procedure called to realize a Stub widget instance.

The procedure has the following declaration:

```
void realize(
    Widget                  w,
    XtValueMask             *value_mask,
    XSetWindowAttributes    *attributes);
```

If the application supplies its own realize procedure, it is the application's responsibility to deal with the `XtNwindow` resource. When `XtNwindow` is non-NULL, the realize procedure uses this window for the widget instance instead of creating a new window. The default realize procedure gives an error message if another widget in its process space is referencing the window already; it does not reparent the specified window.

XtNreferenceStub

Class	Type	Default	Access
XtCReferenceStub	Widget	NULL	GI

Synopsis: The pointer to an existing Stub widget.

If this pointer is non-NULL, the new Stub will inherit all instance methods from the referenced Stub widget. An `XtSetValues()` request on the new Stub widget should be used to change any inherited methods.

XtNregisterFocusFunc

Class	Type	Default	Access
XtCRegisterFocusFunc	Function	NULL	SGI

Synopsis: The procedure called whenever a Stub widget gains focus.

The procedure has the following declaration:

```
Widget registerFocus(Widget stub_widget);
```

Whenever a Stub widget gains focus, this procedure is called and the Stub's shell sets the "current focus widget" (see "OlGetCurrentFocusWidget" on page 151) to the value returned by it. If this function is NULL or returns NULL, the current focus widget is set to the Stub widget. This is the typical case. If this procedure returns a widget ID other than the Stub widget's, that ID is used to update the current focus widget so that a subsequent call to `OlGetCurrentFocusWidget()` would return it. Returning a widget ID other than the Stub widget's will not move the focus away from the Stub widget.

XtNresize

Class	Type	Default	Access
XtCResize	Function	NULL	SGI

Synopsis: The procedure called whenever a Stub widget instance is resized.

The procedure has the following declaration:

```
void resize(Widget w);
```

XtNsetValues

Class	Type	Default	Access
XtCSetValues	Function	NULL	SGI

Synopsis: The procedure called whenever the application makes an `XtSetValues()` call on a Stub widget instance.

The procedure has the following declaration:

```
Boolean setValues(
    Widget          current,
    Widget          request,
    Widget          new,
    ArgList         args,
    Cardinal        *num_args);
```

XtNsetValuesAlmost

Class	Type	Default	Access
XtCSetValuesAlmost	Function	(superclass)	SGI

Synopsis: The procedure called when a parent rejects the requested geometry.

The procedure has the following declaration:

```
void setValuesAlmost(
    Widget              w,
    Widget              new_widget,
    XtWidgetGeometry    *request,
    XtWidgetGeometry    *reply);
```

This procedure is called when the application attempts to set a Stub widget's geometry via an `XtSetValues()` call and the Stub widget's parent did not accept the requested geometry. The default *setValuesAlmost* procedure simply accepts the suggested compromise.

XtNsetValuesHook

Class	Type	Default	Access
XtCSetValuesHook	Function	NULL	SGI

Synopsis: The procedure called whenever the application makes an `XtSetValues()` call on a Stub widget instance.

The procedure has the following declaration:

```
Boolean setValuesHook(
    Widget      w,
    ArgList     args,
    Cardinal    *num_args);
```

Since this procedure is called after the *setValues* procedure, the widget specified by the *w* argument is the *new* widget from the *setValues* procedure.

XtNtraversalHandlerFunc

Class	Type	Default	Access
XtCTraversalHandlerFunc	Function	NULL	SGI

Synopsis: The procedure called to process traversal commands whenever the Stub widget has focus.

If an application wants the Stub widget to process traversal commands whenever the Stub widget has focus, this resource is used to supply the

traversal routine. An example of a case when this is desirable is when a Stub widget is used to implement a spreadsheet. In this case, the Stub widget would trap the OL_MOVERIGHT, OL_MOVELEFT, etc. commands to move focus between the cells in the spreadsheet. The traversal handling routine has the following declaration:

```
Widget traversalHandler(
    Widget          w,
    Widget          start,
    OlVirtualName   direction,
    Time            time);
```

If the traversal routine can process the traversal command, it returns the ID of the widget that now has focus. The widget ID returned can be the Stub widget's ID. This is the case when the traversal command was processed, but focus did not leave the Stub widget. If the traversal routine cannot process the given command, it should return NULL.

XtNwidth

Class	Type	Default	Access
XtCWidth	Dimension	0	SGI

Synopsis: The width of the widget.

If `XtNwindow` has a NULL value, the application must ensure that the dimensions of `XtNwidth` and `XtNheight` are non-NULL. The application can specify the width and height with an Arg list or specify an *initialize* procedure that sets them with non-NULL values. If either of these dimensions is NULL when the application attempts to realize the Stub widget, an error will result.

XtNwindow

Class	Type	Default	Access
XtCWindow	Window	NULL	GI

Synopsis: The window ID that the Stub widget should associate with its instance data at realization time.

The `XtNwindow` resource can be specified at initialization time only. If a window ID is supplied, that Stub widget instance will trap events on the given window. After the Stub widget instance is realized, the function `XtWindow()` will return this window ID. If the Stub widget is managed by its parent widget, the supplied window will be included in geometry calculations even though the Stub widget (by default) does not reparent the supplied window to be a

child of the parent widget's window. Explicit calls to `XtMoveWidget()`, `XtResizeWidget()`, `XtConfigureWidget()`, or `XtSetValues()` can be used to modify the window's attributes.

Note – When the Stub widget instance is destroyed, the window will be destroyed along with it.

Activation Types

The following table lists the activation types used by the Stub.

Table 9-40 Stub Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_HELP	HELP	XtNhelpKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey

The Stub widget has no activation types besides the ones in "Common Activation Types" on page 68.

Example

The following example illustrates how an application can use the Stub widget to perform a particular type of exposure handling. Since an *initialize* procedure was not specified and the `XtNwindow` resource was not used, the initial Arg list includes non-NULL values for the widget's width and height.

```
static void Redisplay(
    Widget      w;
    XEvent      *xevent;
    Region      region);
{           /* do something interesting here */
} /* END OF Redisplay() */

main(...)
{
    Widget      base;
    Widget      stub;
    static Arg args[] = {
        { XtNexpose,    (XtArgVal) Redisplay },
        { XtNwidth,     (XtArgVal) 1 },
        { XtNheight,    (XtArgVal) 1 }
    };

    OlToolkitInitialize();
    base = XtAppInitialize(...);
    stub = XtCreateManagedWidget("graphics pane", stubWidgetClass,
            base, args, XtNumber(args));
    ...
} /* END of main() */
```

See Also

"Input Focus Functions" on page 150.